Exploration of Hotel Guest Ratings Data Through Machine Learning

Reviewed 3 days ago

Fabulous Group Stay

I highly recommend Hotel Emblem to anyone looking for a high-quality stay in SF with a great location and super helpful staff. I planned a holiday party in SF and had about 20 employees who needed rooms and support in terms of holding luggage. Hotel... More

Date of stay: January 2019

"Horrible bed"

This place was nice. We really like the shampoo/conditioner dispenser in the shower. So convenient & neat. We had 2 double beds. One was super comfy & the other one sinked in & made terrible creaking noise with each move. That was the worst. Breakfast to me was GREAT. We love Hotel's hot breakfast & we were super pleased. The staff/single lady took care of the breakfast as in refilling, stocking the breakfast food & materials. Loved her!

Reviewed January 28, 2019

5.0 **Excellent**

This is a gem of a hotel in a wonderful neighborhood. I heartily reccommend it.

7/18/2018

I made a reservation here a couple weeks ago and I paid for my room and went upstairs. I went in there only to find out that my room did not have a bathroom, my closet had a sink in it and that I would have to shower in the hallway and use the restroom in the hallway as well. So I'm like alright it's just one night I'm going to go use the bathroom.

Walking down the hall to urinate the door said vacant and upon opening it a man was in there who immediately shouted that he was in there, appearing to be currently defecating. Upon apologizing to the man, attempting to push the door closed, it appeared to be stuck. I asked the man: "how do you close the door" he replied "I dont know I thought it was locked!" Before proceeding to rise up from the throne, pulling the door shut. I was very disgusted by it and decided to leave. I am convinced this hotel was a brothel at some point or a boarding house similar to hey Arnold. The lady was nice though.

Reviewed 3 days ago

Fabulous Group Stay

I highly recommend Hotel Emblem to anyone looking for a high-quality stay in SF with a great location and super helpful staff. I planned a holiday party in SF and had about 20 employees who needed rooms and support in terms of holding luggage. Hotel... More

Date of stay: January 2019

"Horrible bed"

This place was nice. We really like the shampoo/conditioner dispenser in the shower. So convenient & neat. We had 2 double beds. One was super comfy & the other one sinked in & made terrible creaking noise with each move. That was the worst. Breakfast to me was GREAT. We love Hotel's hot breakfast & we were super pleased. The staff/single lady took care of the breakfast as in refilling, stocking the breakfast food & materials. Loved her!

Reviewed January 28, 2019

5.0 **Excellent**

This is a gem of a hotel in a wonderful neighborhood. I heartily reccommend it.

7/18/2018

I made a reservation here a couple weeks ago and I paid for my room and went upstairs. I went in there only to find out that my room did not have a bathroom, my closet had a sink in it and that I would have to shower in the hallway and use the restroom in the hallway as well. So I'm like alright it's just one night I'm going to go use the bathroom.

Walking down the hall to urinate the door said vacant and upon opening it a man was in there who immediately shouted that he was in there, appearing to be currently defecating. Upon apologizing to the man, attempting to push the door closed, it appeared to be stuck. I asked the man: "how do you close the door" he replied "I dont know I thought it was locked!" Before proceeding to rise up from the throne, pulling the door shut. I was very disgusted by it and decided to leave. I am convinced this hotel was a brothel at some point or a boarding house similar to hey Arnold. The lady was nice though.

Questions

Do hotel features influence the ratings guests give to hotels in review websites?

What words in hotel guest reviews are associated with the ratings they gave in hotel websites?

Machine Learning Objectives

To assess the capacity of different regression models in predicting hotel review ratings based on hotel features.

To obtain sentiment insights from the guest ratings and comments through natural language processing and use these to make predictions.

Data

Dataset	Website
Hotel name, features, location, ratings, comments ● >1,800 hotels ● 10,000 comments	https://data.world
USA airport geographic coordinates ● >13,000 airports	https://opendata.socrata.com/dataset/Airport-Codes-mapped-to-Latitude-Longitude-in-the-/rxrh-4cxm
USA State populations	https://www.census.gov/newsroom/press-kits/2018/pop-estimates-national-state.html

Data was divided into two*

Metadata	Ratings
● Hotel name ● Amenities (e.g., beach, casino, spa, pet-friendly) ● Location	● Hotel names ● Ratings ● Review (i.e., title, text) ● Review date

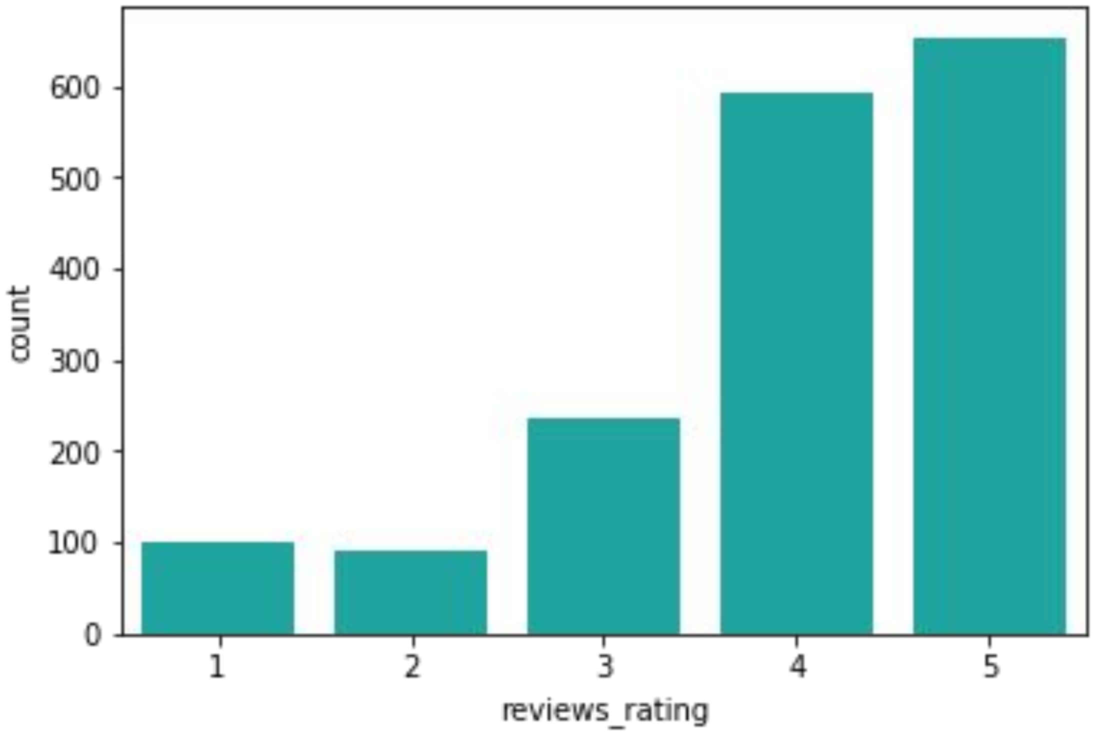
* Data was stored in a SQLite database

Data was extracted from a SQLite database

```python
# Dependencies
import pandas as pd
import sqlite3
```

```python
# Connect to a sqlite database
conn = sqlite3.connect("Data/Hotels.db")

# Get the data from alldata table
alldata = pd.read_sql_query("select * from alldata;", conn)
conn.close()

# Preview the dataframe
alldata.head()
```

Metadata: How are features associated with ratings?



Relationship between state population and number of reviews

```
# Without a constant
import statsmodels.api as sm
```

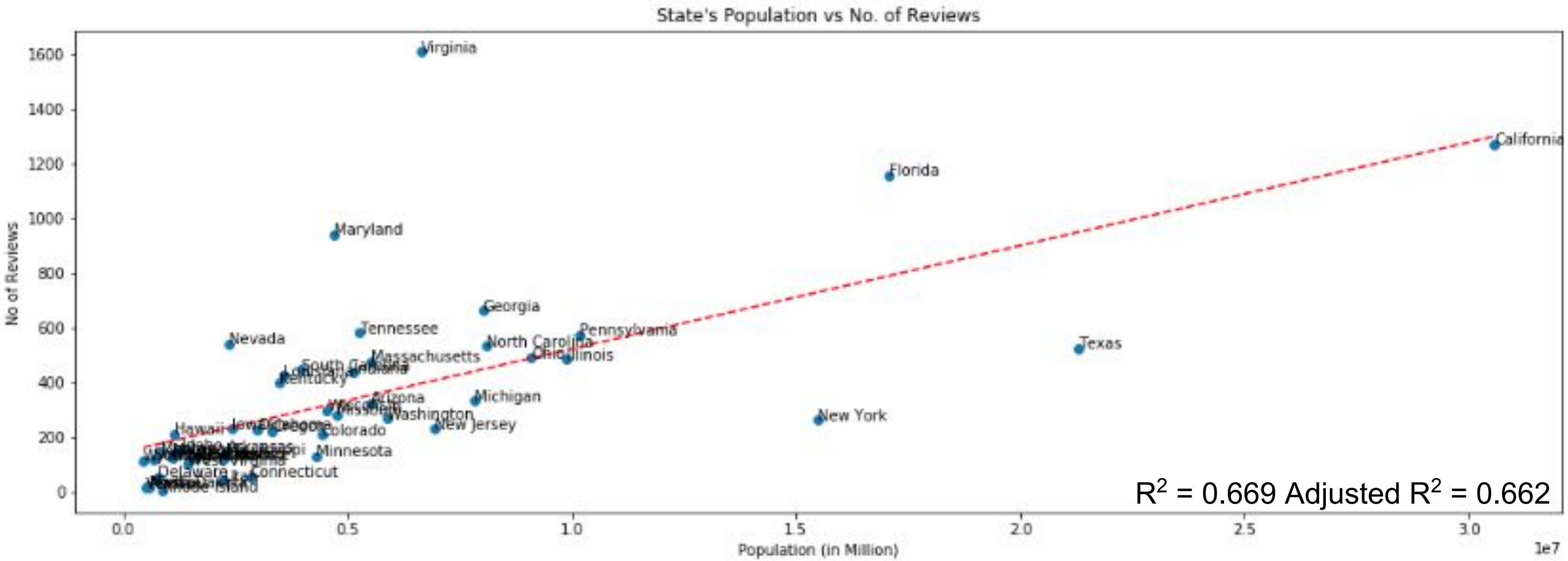
```
# Note the difference in argument order
model = sm.OLS(ratings, Popn_data).fit()
predictions = model.predict(Popn_data)
```

```
# Print out the statistics
model.summary()
```

Relationship between state population and number of reviews

```
x = Popn_data
y = ratings
w = state_data
plt.figure(figsize=(18,6))
plt.scatter(x, y)

for i, txt in enumerate(w):
    plt.annotate(txt, (x[i], y[i]))


z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x,p(x),"r--")


# Create a title, x label, and y label for our chart
plt.title("State's Population vs No. of Reviews")
plt.xlabel("Population (in Million)")
plt.ylabel("No of Reviews")



plt.show()
```

Metadata: State population and guest reviews



Metadata: Feature Addition

Hotel distance from nearest airport (km)

- Calculated using haversine formula (Earth is spherical)

$$d = 2r \arcsin \sqrt{0.5 - \frac{\cos\left((\varphi_2 - \varphi_1)\frac{\pi}{180}\right)}{2} + \frac{\cos\left(\varphi_1 \frac{\pi}{180}\right)\cos\left(\varphi_2 \frac{\pi}{180}\right)\left(1 - \cos\left((\lambda_2 - \lambda_1)\frac{\pi}{180}\right)\right)}{2}}$$

- For each hotel coordinate, the distances for all airports were calculated and the min distance was recorded

Metadata: Feature Addition

```
# Dependencies for calculating distances on Earth (sphere) using haversine formula
# Source2: https://stackoverflow.com/a/21623206
from math import cos, asin, sqrt, pi

def distance(lat1, lon1, lat2, lon2):
    """distance is expressed in km"""
    p = pi/180 #factor to convert degrees to radians
    a = 0.5 - cos((lat2-lat1)*p)/2 + cos(lat1*p)*cos(lat2*p) * (1-cos((lon2-lon1)*p)) / 2
    return 12742 * asin(sqrt(a)) # Earth diameter: 12742 = 2 * R; R = 6371km (mean radius of the earth)

# Find the minimum distance between the hotel and the closest airport
def min_distance(lat, lon):
    distances = []
    for i in range(0, len(airports)):
        away = distance(abs(lat), abs(lon), abs(airports["Latitude"][i]), abs(airports["Longitude"][i]))
        distances.append(away)
    return min(distances)
```

```
# Get latitudes and longitudes of airports mapped in the USA
# Source: https://opendata.socrata.com/dataset/Airport-Codes-mapped-to-Latitude-Longitude-in-the-/rxrh-4cxm
path = "Data/Airport_Codes_Coords_USA.csv"

airports = pd.read_csv(path)
airports.head()
```

Metadata: Feature Addition

```
# For each hotel coordinate, calculate the distance to the nearest airport
airport_distance = []
for i in range(0, len(df4)):
    dist = min_distance(df4["latitude"][i], df4["longitude"][i])
    airport_distance.append(dist)

    print(f"Now processing {i}th airport.\n-----")
    clear_output(wait = True) # to replace output with new one (instead of printing many outputs)
```

```
Now processing 1852th airport.
-----
```

Metadata: Dummy variables

Tokenisation

For each review, the hotel categories/description (string) was converted to a list of strings

Hotel feature selection

Lists for all hotels merged and hotel features were selected manually (list)

Hotel feature to dummy variable

For each hotel's list of strings: if a hotel feature is present, feature = 1; else, feature = 0

State to dummy variable

If hotel location (state) is the state feature, state feature = 1; else state feature = 0

Metadata: Dummy variables

```
# Create new columns containing categories/features for the hotels (manually pick from unique terms)
features = ['resort', 'movie', 'marina', 'harbor', 'reception',
            'convention', 'health', 'hall', 'extended', 'lodges',
            'chalets', 'cemetery', 'skiing', 'theater', 'campground',
            'entertainment', 'clinics', 'cabins', 'parties', 'lodge',
            'nightclub', 'services', 'airport', 'spas', 'hotel', 'pools',
            'attractions', 'meeting', 'utility', 'condominiums', 'cable',
            'office', 'village', 'fashion', 'loft', 'chapels', 'fairgrounds',
            'boutique', 'gym', 'motel', 'bars', 'e-commerce',
            'golf', 'apartment', 'medical', 'pubs', 'cottages', 'pet', 'lakeview',
            'restaurant', 'wedding', 'fitness', 'recreation', 'receptions',
            'reservations', 'casino', 'family-friendly', 'breakfast',
            'beach', 'karaoke']

# Sort the items in the list alphabetically
features.sort()

for feat in features:
    df4[feat] = np.nan
```

```
# Create a function that fills in 1s and 0s for selected categories
def Cat_encoding(category):
    for i in range(0,len(df4)):
        if category not in df4["categories2"][i]:
            df4[str(category)][i] = "0"
        else:
            df4[str(category)][i] = "1"

        print(f"Now processing {i}th hotel for {category}.\n-----")
        clear_output(wait = True) # to replace output with new one (instead of printing many outputs)
```

Metadata: Feature selection

Addressing multicollinearity

- Calculated correlation coefficients (φ) for binary variables

$$\varphi = \frac{(n_{11}n_{00}) - (n_{10}n_{01})}{\sqrt{n_{1\cdot}n_{0\cdot}n_{\cdot 0}n_{\cdot 1}}}$$

- For feature pairs with r ≥ 0.70, one feature was excluded from the analysis

Metadata: Feature selection

```
# from sklearn import matthews_corrcoef (calculates phi coefficient)
from sklearn.metrics import matthews_corrcoef

from IPython.core.display import clear_output

var_pairs = [(x, y) for x in df3.columns.values for y in df3.columns.values]

phi_list = []
for f in var_pairs:
    phi = matthews_corrcoef(list(df3[f[0]]), list(df3[f[1]]))
    phi_list.append(phi)

    print(f"Calculating phi coefficients for {f}.")
    print("---")
    clear_output(wait = True)
```

```
Calculating phi coefficients for ('2018', '2018').
---
```

Metadata: Feature selection

```
# Create a dataframe that contains tuples of variable pairs and phi coefficients
df5 = pd.DataFrame({"pair": var_pairs, "phi": phi_list})

# Split the tuples and put each variable in separate columns
df5[["var1", "var2"]] = pd.DataFrame(df5["pair"].tolist())

# Preview the dataframe
df5.head()
```

```
# What variables to remove from the analyses?
# Basis: correlation coefficient, r, is higher than 0.69 or lower than -0.69

high = []
for x in range(0, len(df5)):
    if df5["phi"][x] != 1 and df5["phi"][x] >= 0.7 or df5["phi"][x] <= -0.7:
        high.append(df5["pair"][x])
print(high)
```

```
[('hall', 'nightclub'), ('motel', 'reservations'), ('nightclub', 'hall'), ('reservations', 'm
otel')]
```

```
# Remove a variable that is highly correlated with the other from dataframe df2
df4 = df2.drop(columns = ["hall", "motel"], axis = 1)
df4.head()
```

Metadata: Feature selection

Addressing multicollinearity

- Lasso and Ridge Regression models
- Increased alpha values decreased coefficients to 0 (thereby reducing features)

Metadata: Feature selection

Addressing near-zero and zero-variance features

- Applied a variance thresholder to exclude features that had very few, or no, different values
- X[“hotels”] = [1,1,1,1,1,…, 1]

  ∴ X[“hotels”] was excluded from the model

Metadata: Feature selection

```
# Define the response (y) and the explanatory (X) variables
X = df.drop(columns = ["reviews_rating"], axis = 1)
y = df["reviews_rating"]

# Standardise the explanatory variables
scaler = StandardScaler()
X_standard = scaler.fit_transform(X)

# Convert review rating into whole number
y = y.astype(int)

print(X_standard.shape, y.shape)
```

```
(1670, 124) (1670,)
```

```
# Create a thresholder model
thresholder = VarianceThreshold(threshold = 0.5)

# Conduct variance thresholding
Xstd_high_var = thresholder.fit_transform(X_standard)

print(f"Number of variables before thresholding: {df.shape[1]}")
print(f"Number of variables after thresholding: {len(Xstd_high_var[0])}")
```

Metadata: Feature selection

Random forest

- Used to calculate variable importance for 121 features so that only the 10 most important variables are included in regression analyses

Metadata: Feature selection

```
# Dependencies
import mord
from sklearn.ensemble import RandomForestRegressor
```

```
# Create a random forest regressor (for continuous explanatory variables, like scaled values)
rf = RandomForestRegressor(n_estimators = 500)
rf_reg = rf.fit(Xstd_high_var, y)

# Get the coefficient of determination (R^2) of the random forest prediction
rf_reg.score(Xstd_high_var, y)

print(f"R^2 using random forest: {rf_reg.score(Xstd_high_var, y)}")
```

```
R^2 using random forest: 0.7606791518932264
```

```
# Variable importance
impt = rf_reg.feature_importances_
impt_var = sorted(zip(impt, list(X.columns)), reverse = True)[0:10]
impt_var
```

Metadata: Selected features

Features	Importance
Nearest airport distance	0.29
Reservations	0.04
2013	0.04
California	0.03
2011	0.03

Features	Importance
2012	0.02
Family-friendly	0.02
Florida	0.02
2010	0.02
Texas	0.02

Metadata: Regression Analyses

Regression model	Library	Description
Multinomial logistic regression	sklearn	Multiclass categorical response variable
Ridge regression	sklearn	Prevents multicollinearity by shrinking the coefficients
Ordinal logistic regression (AT)	mord	Ranked categorical response variable
Ordinal logistic regression (IT)	mord	Ranked categorical response variable
Lasso regression	sklearn	Prevents multicollinearity by shrinking the coefficients

Metadata: Regression analyses

General workflow

1. Data was split into training and testing datasets
2. Models were defined
3. Models were fitted using the training data
4. Predictions were made using the testing data
5. Model performances were evaluated using accuracy, error, R^2, and f1 score

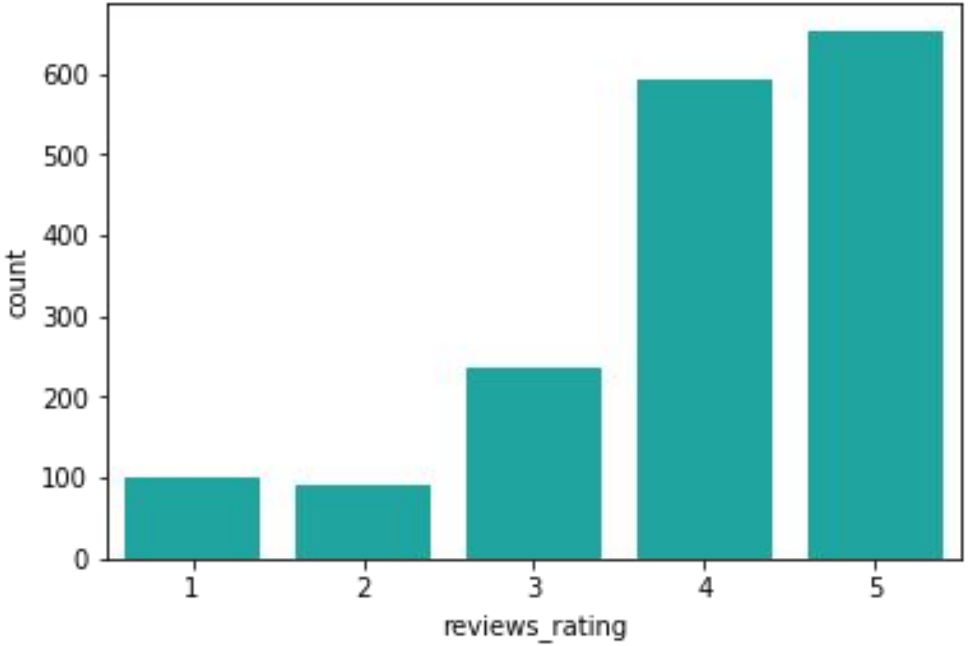
Metadata: Regression Analyses

Regression model	Mean abs error	Training mean accuracy	Testing mean accuracy	R^2	Avg f1-score
Multinomial logistic regression	1.00	0.40	0.40	-0.74	0.32
Ridge regression	0.86	0.06	-0.01	-0.01	N/A
Ordinal logistic regression (AT)	0.83	-0.81	-0.83	-0.01	0.20
Ordinal logistic regression (IT)	1.07	0.40	0.39	-0.93	0.27
Lasso regression	0.86	0.00	-0.00	-0.01	N/A

Metadata: What the analyses suggest

- The regression models have low accuracy & f1-scores
- The models have low predictive ability
- Models may be limited by mostly dummy variables
- Perhaps, using continuous variables will improve the models (but accessibility is limited)
- Predictability might be improved if the distribution of ratings was not skewed to the left

Ratings: Overview



Many guest reviews indicated high scores

- Mostly loved their hotel experience?
- Didn't write about bad experiences?
- Social desirability bias?

Ratings: Natural Language Processing

Data Cleaning

In [1]:
```
# Dependencies
import pandas as pd
import numpy as np
import sqlite3
from pyspark import SparkContext
sc =SparkContext()
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
import collections
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Create connections to database
conn = sqlite3.connect("Data/Hotels.db")

#Load the database table into a pandas dataframe
ratings = pd.read_sql_query("select * from ratings;", conn)
conn.close()

# Preview the dataframe
ratings.head()
```

Out[2]:

	index	name	reviews_date	reviews_rating	reviews_sourceURLs	reviews_text	reviews_title	reviews_userCity	reviews_userProvince
0	0	Rancho Valencia Resort Spa	2013-11-14T00:00:00Z	5.0	www.hotels.com	Our experience at Rancho Valencia was absolute...	Best romantic vacation ever!!!!	None	None
1	1	Rancho Valencia Resort Spa	2014-07-06T00:00:00Z	5.0	www.hotels.com	Amazing place. Everyone was extremely warm and...	Sweet sweet serenity	None	None
2	2	Rancho Valencia Resort Spa	2015-01-02T00:00:00Z	5.0	www.hotels.com	We booked a 3 night stay at Rancho Valencia to...	Amazing Property and Experience	None	None
3	3	Aloft Arundel Mills	2016-05-15T00:00:00Z	2.0	www.tripadvisor.com	Currently in bed writing this for the past hr ...	Never again...beware, if you want sleep.	Richmond	VA
4	4	Aloft Arundel Mills	2016-07-09T00:00:00Z	5.0	www.tripadvisor.com	I live in Md and the Aloft is my Home away fro...	ALWAYS GREAT STAY...	Laurel	MD

click to scroll output; double click to hide

- Created a model that predicts hotel ratings based on guest comments:

- Dependencies were imported.
- Data was extracted from a SQLite database.

NLP: Data Cleaning

```
In [4]: #Grouping ratings to get 5 unique ratings
rating_list = ratings['reviews_rating'].tolist()
new_list = []
for rating in rating_list:
    if rating >= 5.0:
        new_list.append(5.0)
    elif rating >= 4 and rating < 5:
        new_list.append(4.0)
    elif rating >= 3 and rating < 4:
        new_list.append(3.0)
    elif rating >= 2 and rating < 3:
        new_list.append(2.0)
    else:
        new_list.append(1.0)
```

```
In [5]: # Put all letters in lower case
# Split hotel reviews_rating to "good"/"bad"
ratings["reviews_text"] = ratings["reviews_text"].str.lower()
ratings['reviews_text'] = ratings['reviews_text'].astype(str)
ratings["rating"] = new_list
ratings.head()
```

Out[5]:

	index	name	reviews_date	reviews_rating	reviews_sourceURLs	reviews_text	reviews_title	reviews_userCity	reviews_userProvince	rating
0	0	Rancho Valencia Resort Spa	2013-11-14T00:00:00Z	5.0	www.hotels.com	our experience at rancho valencia was absolute...	Best romantic vacation ever!!!!	None	None	5.0

```
In [7]: words_list = []
preprocessed_text = []
for review in ratings["reviews_text"]:

    # Create a list of words per rating after the words are converted to lowercase
    words = word_tokenize(review)

    # Filter to remove stop words and punctuations
    words2 = [word for word in words if word not in stops and word not in exclude]

    # Add the filtered list of words
    words_list.append(words2)

    # Convert the list of strings back to one string
    words3 = " ".join(words2)

    # Add the filtered list of words
    preprocessed_text.append(words3)

ratings["filteredReview"] = preprocessed_text
```

```
In [8]: # Remove columns that will not be used in the analysis
df = ratings.drop(columns=["reviews_text","reviews_date","reviews_sourceURLs","reviews_title","reviews_userCity","review
df.head()
```

Out[8]:

	index	name	reviews_rating	rating	filteredReview
0	0	Rancho Valencia Resort Spa	5.0	5.0	experience rancho valencia absolutely perfect ...
1	1	Rancho Valencia Resort Spa	5.0	5.0	amazing place everyone extremely warm welcomin...
2	2	Rancho Valencia Resort Spa	5.0	5.0	booked 3 night stay rancho valencia play tenni...
3	3	Aloft Arundel Mills	2.0	2.0	currently bed writing past hr 1/2 dogs barking...
4	4	Aloft Arundel Mills	5.0	5.0	live md aloft home away home ... stayed 1 nigh...

Step 1

Data was processed and grouped into 5 unique ratings.

Step 2

Text was converted into lowercase and data type was changed to type string. Column with new ratings was added to DataFrame.

Step 3

Data was filtered to remove stop words and punctuations

Step 4

Columns that won't be use for further analyses were removed from original DataFrame

NLP: Process

01	**Pandas DataFrame was converted into Spark DataFrame**	• Column “length” was created to show display number of words in review
02	**Dependencies were imported and all features was created to the data set**	• Tokenize by using Tokenizer() • Remove Stop Words by using StopWordsRemover() • Hashing the Data by using HashingTF() • Create IDF token by using IDF()
03	**Create feature vectors and Pipeline**	• Create feature vectors by using VectorAssembler() • Create a data processing Pipeline by using Pipeline()
04	**Fit and transform the pipeline**	• Fit Pipeline • Transform Pipeline
05	**Split Data into Training/Testing and make a prediction**	• Break data down into a training set and a testing set • Transform the model with the testing data(make a prediction) • Evaluate Accuracy of model

NLP: Process

In [9]:
```
#Convert Pandas DataFrame to Spark DataFrame
spark_ratings = sqlContext.createDataFrame(df)
spark_ratings.show(3)
```

```
+-----+--------------------+--------------+------+--------------------+
|index|                name|reviews_rating|rating|      filteredReview|
+-----+--------------------+--------------+------+--------------------+
|    0|Rancho Valencia R...|           5.0|   5.0|experience rancho...|
|    1|Rancho Valencia R...|           5.0|   5.0|amazing place eve...|
|    2|Rancho Valencia R...|           5.0|   5.0|booked 3 night st...|
+-----+--------------------+--------------+------+--------------------+
only showing top 3 rows
```

1

In [10]:
```
# Create a length column to be used as a future feature
from pyspark.sql.functions import length
data = spark_ratings.withColumn('length', length(spark_ratings['filteredReview']))
data.show(3)
```

In [11]:
```
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF, StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vector
from pyspark.ml import Pipeline
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

2

In [12]:
```
# Create all the features to the data set
pos_neg_to_num = StringIndexer(inputCol='rating',outputCol='label')
tokenizer = Tokenizer(inputCol="filteredReview", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
hashingTF = HashingTF(inputCol="stop_tokens", outputCol='hash_token',numFeatures=pow(2,4))
idf = IDF(inputCol="hash_token", outputCol="idf_token")
```

NLP: Process

3

```
In [13]: # Create feature vectors
clean_up = VectorAssembler(inputCols=['label','idf_token', 'length'], outputCol='features')
```

```
In [14]: # Create a data processing Pipeline
data_prep_pipeline = Pipeline(stages=[pos_neg_to_num, tokenizer, stopremove, hashingTF, idf, clean_up])
```

4

```
In [15]:  # Fit and transform the pipeline
cleaner = data_prep_pipeline.fit(data)
cleaned = cleaner.transform(data)
cleaned.show(3)
```

```
+-----+--------------------+--------------+------+--------------------+------+-----+--------------------+------------
--------+--------------------+--------------------+--------------------+
|index|                name|reviews_rating|rating|      filteredReview|length|label|          token_text|         sto
p_tokens|          hash_token|           idf_token|            features|
+-----+--------------------+--------------+------+--------------------+------+-----+--------------------+------------
--------+--------------------+--------------------+--------------------+
|    0|Rancho Valencia R...|           5.0|   5.0|experience rancho...|   112|  0.0|[experience, ranc...|[experience,
ranc...|(16,[1,4,5,6,8,9,...|(16,[1,4,5,6,8,9,...|[0.0,0.0,0.184744...|
```

5

```
In [17]: # Break data down into a training set and a testing set
training, testing = cleaned.randomSplit([0.7, 0.3])
```

```
In [18]: # Create a Naive Bayes model and fit training data
nb = NaiveBayes()
predictor = nb.fit(training)
```

```
In [19]: # Tranform the model with the testing data
test_results = predictor.transform(testing)
test_results.show(3)
```

```
+-----+--------------------+--------------+------+--------------------+------+-----+--------------------+------------
--------+--------------------+--------------------+--------------------+--------------------+--------------------+---
--------+
|index|                name|reviews_rating|rating|      filteredReview|length|label|          token_text|         sto
p_tokens|          hash_token|           idf_token|            features|       rawPrediction|         probability|pre
diction|
+-----+--------------------+--------------+------+--------------------+------+-----+--------------------+------------
--------+--------------------+--------------------+--------------------+--------------------+--------------------+---
--------+
|    1|Rancho Valencia R...|           5.0|   5.0|amazing place eve...|   202|  0.0|[amazing, place, ...|[amazing, pl
ace, ...|(16,[2,3,4,5,6,7,...|(16,[2,3,4,5,6,7,...|[0.0,0.0,0.0,0.28...|[-63.563472902385...|[0.79889470223004...|
0.0|
```

NLP: Insights

```
# rating: 5 - 0.0, 4 - 1.0, 3 - 2.0, 2 - 3.0, 1 - 4.0
test_results.select(['label', 'prediction']).show(20)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  3.0|       2.0|
|  3.0|       1.0|
|  1.0|       1.0|
|  3.0|       4.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  0.0|       0.0|
|  2.0|       2.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  2.0|       2.0|
|  0.0|       0.0|
|  1.0|       1.0|
|  3.0|       2.0|
|  2.0|       2.0|
+-----+----------+
only showing top 20 rows
```

```
# Use the Class Evaluator for a cleaner description
acc_eval = MulticlassClassificationEvaluator()
acc = acc_eval.evaluate(test_results)
print("Accuracy of model at predicting ratings was: ", acc)
```

```
Accuracy of model at predicting ratings was:  0.7545075158776297
```

01

Naive Bayes Model was used to make this prediction. This modely is widely used in Text classification/ Spam Filtering/ Sentiment Analysis.

02

In our case the Model shows that the presence of some words in the review lead to a specific rating. However, Naive Bayse Model states that each feature independently contributes to the probability of the result.

03

If others features will be taken into account the Accuracy of the model might be a subject to change(Increase/Decrease)

04

The model has 75% prediction accuracy.

05

As an example, 16 out of 20 ratings match with original ratings given by customers

Ratings: What the analyses suggest

- The model has a relatively high predictive ability
- It can be used to predict the ratings hotel guests would have given (if there was a rating scale) based on their comments
- Can it be used for sentiment analyses for other services?
  - What are the common words?
  - What are the important words, based on term frequency-inverse document frequency?

Most Common Words

- Data was analyzed to find most common words for each rating

Step 1

Guest reviews were grouped by rating and all reviews were combined together as a single text.

Step 2

Stop words and punctuation marks were removed. Unique words were counted.

Step 3

Words and count of words were split into separate lists and combined together in DataFrame with ratings.

Step 4

Common words were visualised:
-Heatmap
- Interactive Graph
- Word Clouds

Most Common Words

```
#Filtering Rating 5.0
filtered_5 = df[df['rating'] == 5.0]
list_filtered_5 = filtered_5["filteredReview"].tolist()
list_filtered_5_new = ''.join(list_filtered_5)
#Filtering Rating 4.0
filtered_4 = df[df['rating'] == 4.0]
list_filtered_4 = filtered_4["filteredReview"].tolist()
list_filtered_4_new = ''.join(list_filtered_4)
#Filtering Rating 3.0
filtered_3 = df[df['rating'] == 3.0]
list_filtered_3 = filtered_3["filteredReview"].tolist()
list_filtered_3_new = ''.join(list_filtered_3)
#Filtering Rating 2.0
filtered_2 = df[df['rating'] == 2.0]
list_filtered_2 = filtered_2["filteredReview"].tolist()
list_filtered_2_new = ''.join(list_filtered_2)
#Filtering Rating 1.0
filtered_1 = df[df['rating'] == 1.0]
list_filtered_1 = filtered_1["filteredReview"].tolist()
list_filtered_1_new = ''.join(list_filtered_1)
```

Most Common Words

```
#Extending stop words list to avoid useless words
stops_hotel = stopwords.words("english")
newStopWords = ['hotel','room','staff','rooms', 'stay', 'stayed' 'breakfast','location', "n't", "", "one", "us", 'desk'
stops_hotel.extend(newStopWords)
```

```
#Counting words for rating 5.0
wordcount5 = {}
for word in list_filtered_5_new.lower().split():
    word = word.replace(".","")
    word = word.replace(",","")
    word = word.replace(":","")
    word = word.replace("\"","")
    word = word.replace("!","")
    word = word.replace("â€œ","")
    word = word.replace("â€˜","")
    word = word.replace("*","")
    if word not in stops_hotel:
        if word not in wordcount5:
            wordcount5[word] = 1
        else:
            wordcount5[word] += 1

sorted_by_value5 = sorted(wordcount5.items(), key=lambda kv: kv[1], reverse=True)[:20]
word_list5 = []
count_list5 = []
for element in sorted_by_value5:
    word_list5.append(element[0])
    count_list5.append(element[1])
```

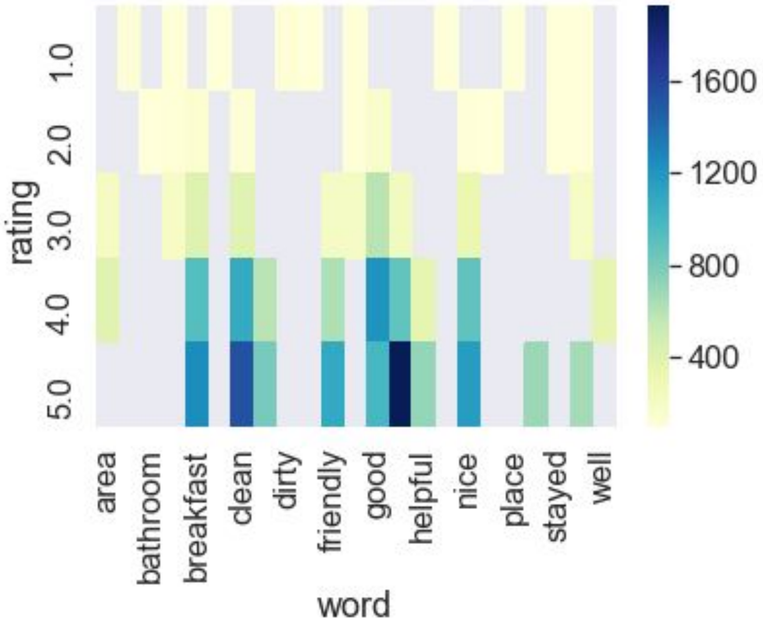
Most Common Words

```
#Creating DF for rating 5.0
df5 = pd.DataFrame(list(zip(word_list5, count_list5)),
                   columns =['word', 'count'])
df5['rating'] = '5.0'
#Creating DF for rating 4.0
df4 = pd.DataFrame(list(zip(word_list4, count_list4)),
                   columns =['word', 'count'])
df4['rating'] = '4.0'
#Creating DF for rating 3.0
df3 = pd.DataFrame(list(zip(word_list3, count_list3)),
                   columns =['word', 'count'])
df3['rating'] = '3.0'
#Creating DF for rating 2.0
df2 = pd.DataFrame(list(zip(word_list2, count_list2)),
                   columns =['word', 'count'])
df2['rating'] = '2.0'
#Creating DF for rating 1.0
df1 = pd.DataFrame(list(zip(word_list1, count_list1)),
                   columns =['word', 'count'])
df1['rating'] = '1.0'
```

```
#Common Data Frame
frames = [df5, df4, df3, df2, df1]
df = pd.concat(frames)
df.head()
```

Associating words with ratings

```
heat_map_data = short_df.pivot("rating",'word','count')
heat_map = sns.heatmap(heat_map_data, cmap="YlGnBu")
```

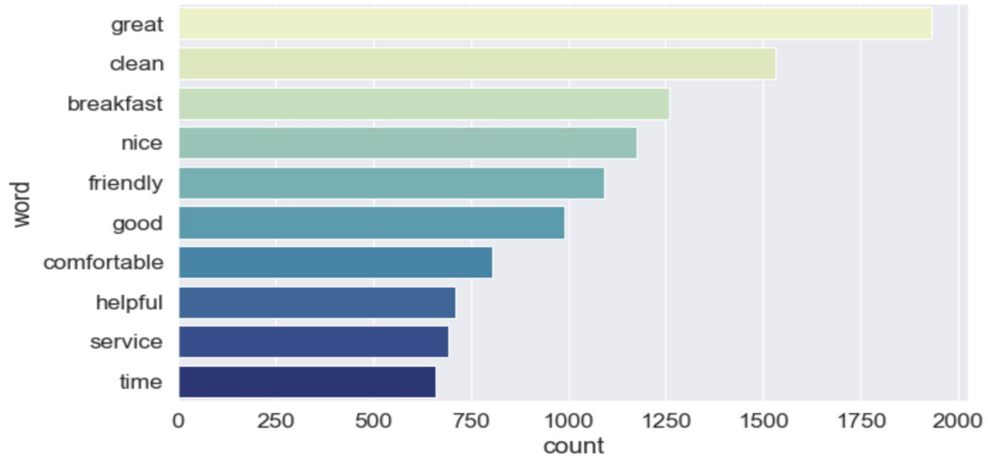


Most Common Words: Visualisation

```
# Create a filter based on title
def plot_it(rating_title):
    if rating_title != "Choose a rating...":
        df1 = short_df[short_df["rating"] == rating_title]

        plt.figure(figsize = (10, 6))
        sns.set(font_scale = 1.5)
        graph = sns.barplot(y = "word", x = "count", data = df1, palette = "YlGnBu")
```

```
# Plot the data by poem title
interactive(plot_it, rating_title = rating_title)
```

rating: 5.0



Most Common Words: Visualisation

```
#world cloud preparation
wc_prep = df["word"].tolist()
wcloud_prep = ' '.join(wc_prep)
# read the mask image
case_mask = np.array(Image.open("images/case.jpg"))
# Make the figure
wordcloud = WordCloud(mask=case_mask, background_color="white", contour_color='steelblue',
                      min_font_size=16).generate(wcloud_prep)
plt.figure(figsize = (15, 6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

Word Cloud for All Ratings

Word Clouds by Guest Rating

Most important words by TF-IDF

General workflow

1. Used the ratings table in the database
2. Merged reviews and titles into one cell
3. Grouped the data based on either:
    a. Hotel name (to find important words for each hotel)
    b. Guest rating (to find important words for each rating)
4. Filter reviews: tokenise, remove stop words, lemmatise
5. Calculate TF-IDF by hotel or by guest rating

Most important words by TF-IDF

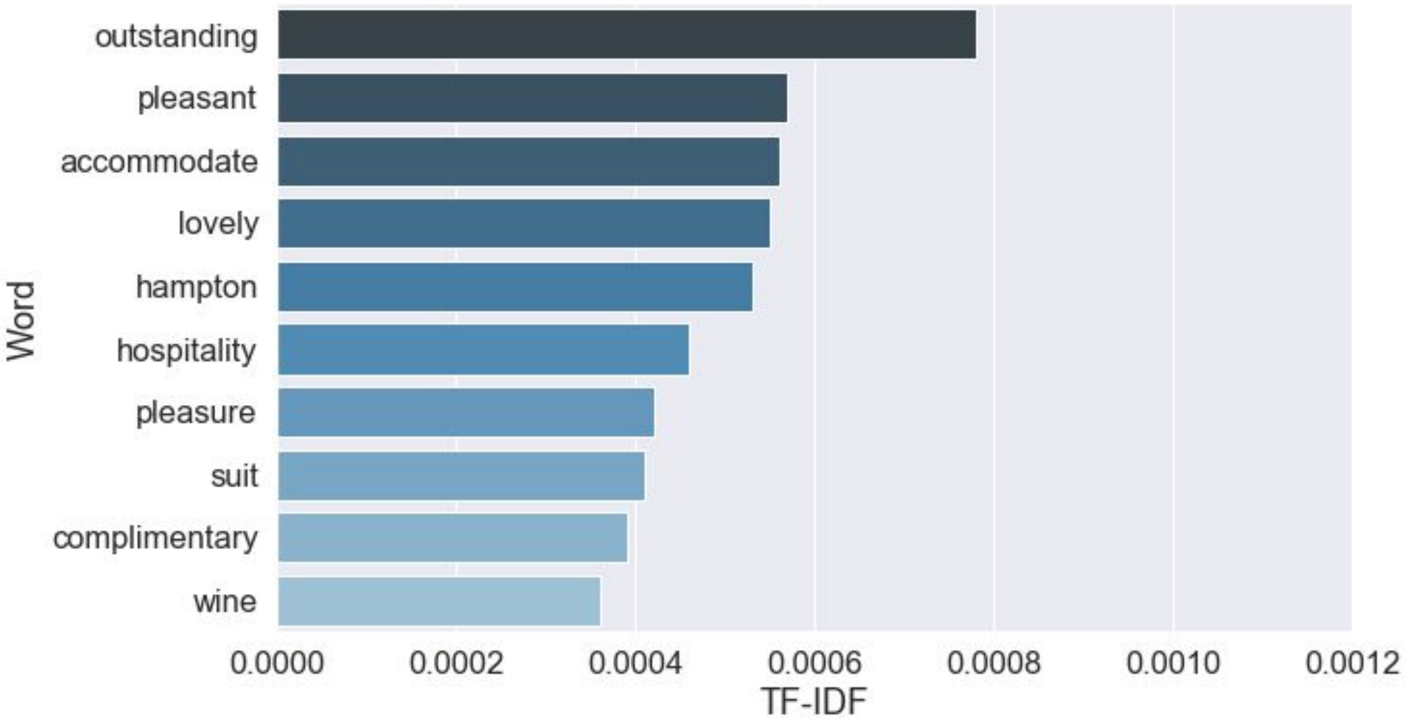
```
# Dependencies
import math
from textblob import TextBlob as tb
```

```
# Create a function that calculates term frequency
def tf(word, review):
    return review.words.count(word) / len(review.words)

# Create a function that determines the number of documents that contain a certain word
def n_docs(word, reviewlist):
    return sum(1 for review in reviewlist if word in review.words)

# Create a function that determines the inverse document frequency (IDF)
# IDF = how common a word is among all the documents in reviewlist
def idf(word, reviewlist):
    return math.log(len(reviewlist) / (1 + n_docs(word, reviewlist)))

def tdidf(word, review, reviewlist):
    return tf(word, review) * idf(word, reviewlist)
```

Most important words by TF-IDF

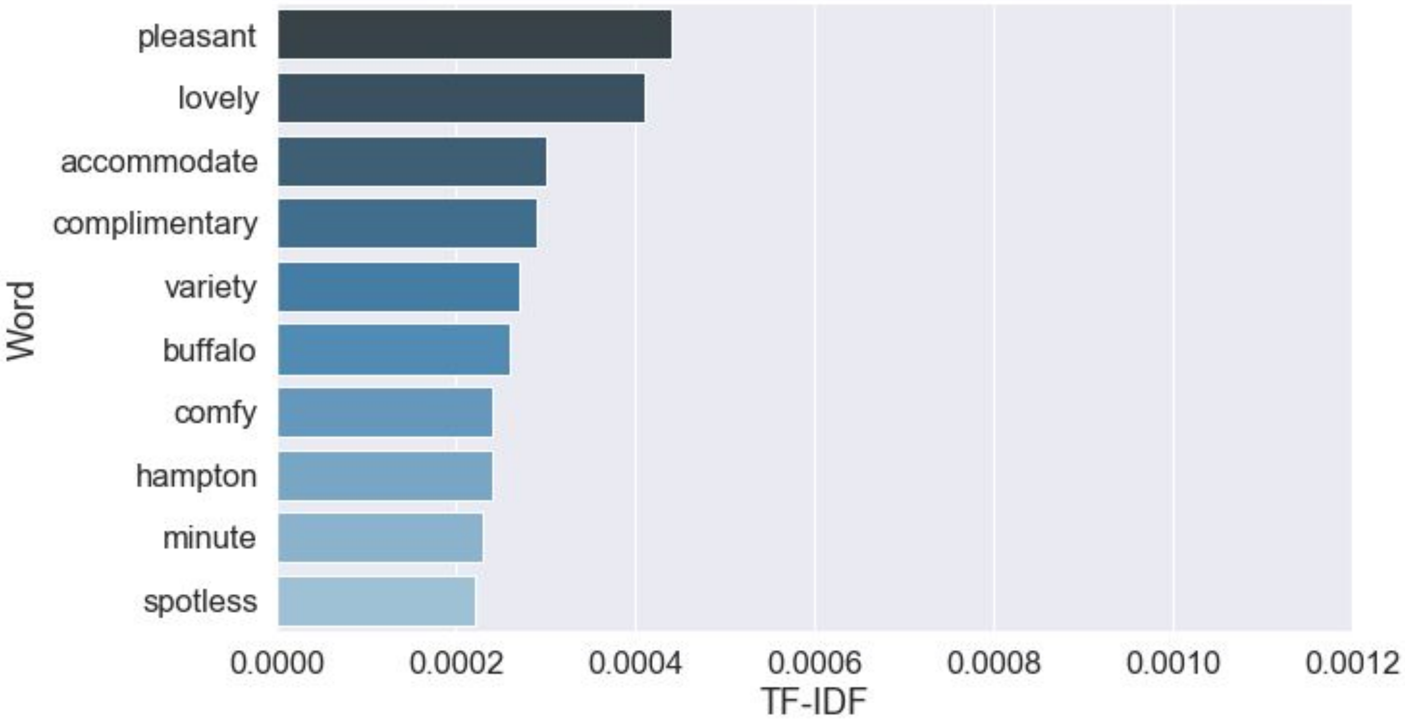
```
# Create a function that calculates the TF-IDF for important words in dataframes
def calc_TFIDF2(df):
    reviewlist = [tb(review) for review in rate_df2["filteredReview"]]

    # Create an empty list to be filled with text blobs from cleaning reviewlist
    reviewlist2 = []

    # Loop through the reviewlist
    for i in range(0, len(reviewlist)):

        # Remove words that are shorter than 3 characters
        new_string = ' '.join([w for w in str(reviewlist[i]).split() if len(w) > 3])

        # Replace em dash and period with space
        new_string2 = new_string.replace("—", " ")
        new_string2 = new_string.replace(".", " ")

        # Convert string to text blob
        new_string2 = tb(new_string2)

        # Append the text blob to the list of text blobs
        reviewlist2.append(new_string2)

    # Calculate the five most important words
    impt_words = []
    for i, review in enumerate(reviewlist2):
        scores = {word: tdidf(word, review, reviewlist2) for word in review.words}
        sorted_words = sorted(scores.items(), key = lambda x: x[1], reverse = True)

        for word, score in sorted_words[:10]:
            impt_words.append((i + 1, word, round(score, 5)))

    # Create a dataframe of important words per review
    df2 = pd.DataFrame(impt_words, columns = ["Rating", "Word", "TF-IDF"])

    return df2
```

Top 10 important words based on TF-IDFs



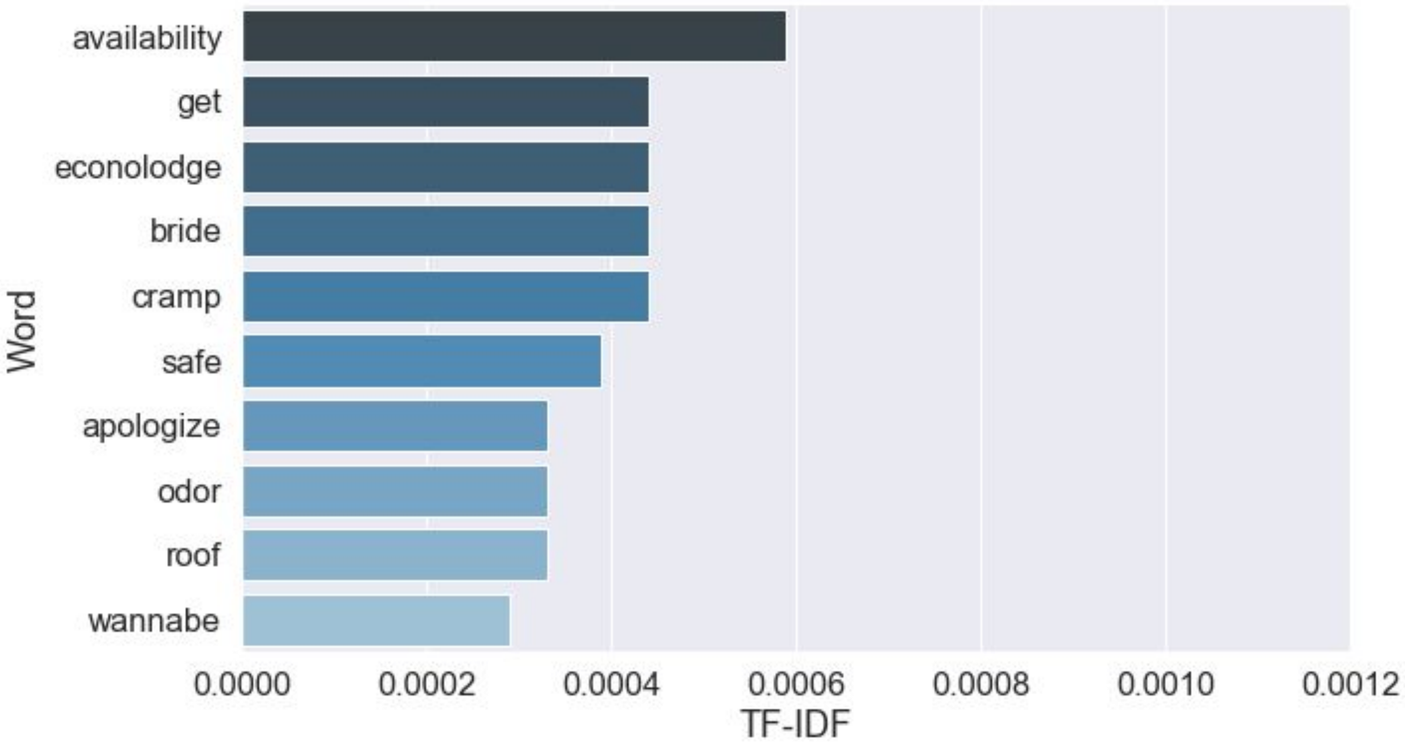
Avg Rating = 5

Top 10 important words based on TF-IDFs



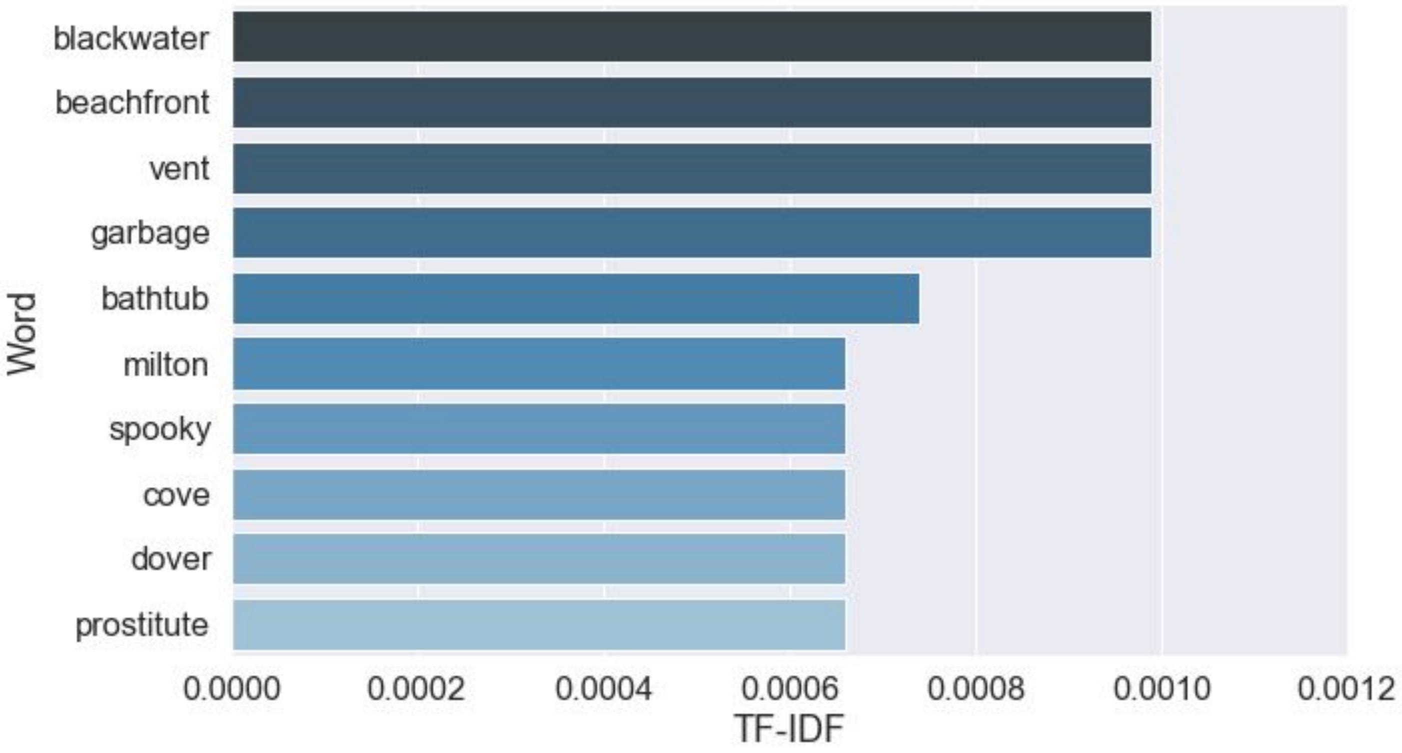
Avg Rating = 4

Top 10 important words based on TF-IDFs



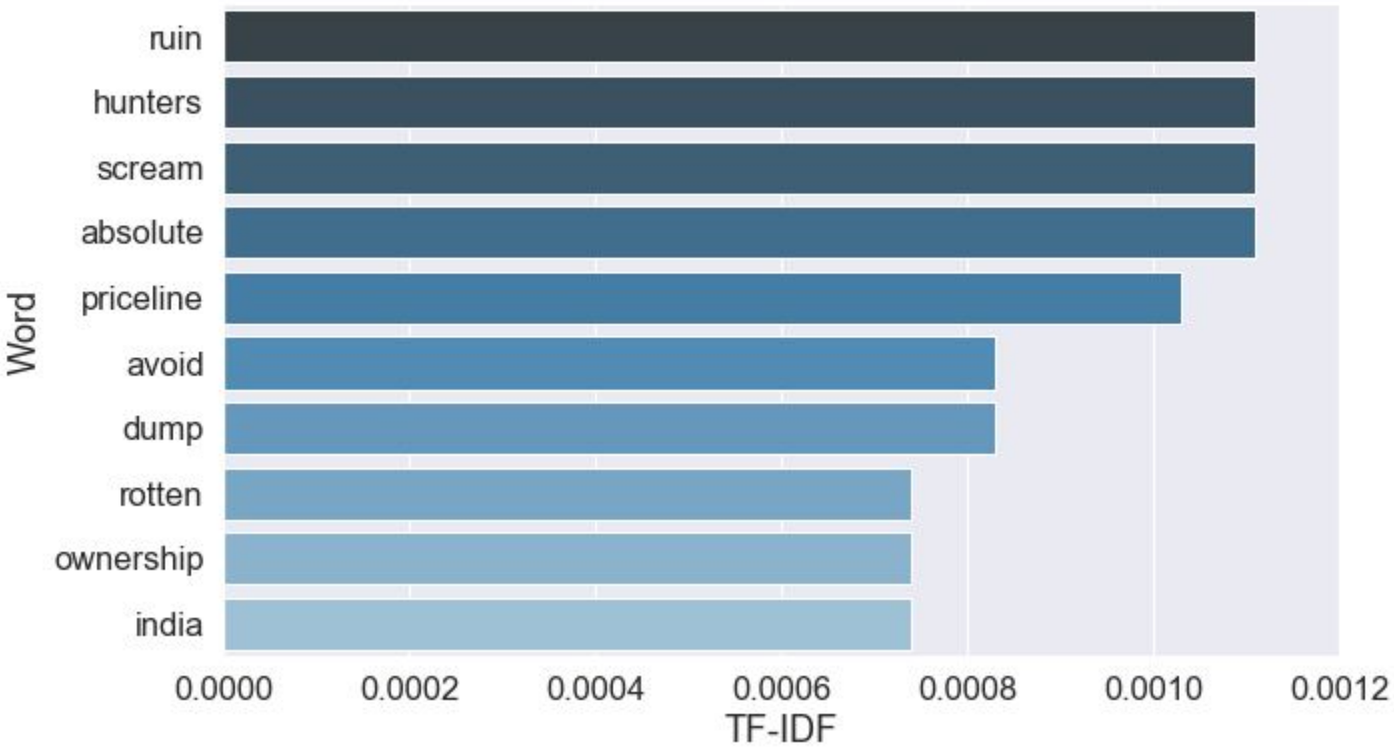
Avg Rating = 3

Top 10 important words based on TF-IDFs



Avg Rating = 2

Top 10 important words based on TF-IDFs



Avg Rating = 1