BACCHANAL BUFFET

A COMPREHENSIVE STUDY USING YELP DATA

AHMET FARUK İZGÖRDÜ

Table Of Content

- Introduction
- Methodology
- Results
- Conclusion

Introduction

Overview of Business Need

This project addresses the need for a systematic approach to assess customer sentiments from Yelp reviews, enabling the restaurant to make data-driven decisions that enhance customer satisfaction and loyalty.

Solution Approach

To address the business need, we have implemented a sentiment analysis model on Yelp reviews using advanced natural language processing (NLP) techniques.

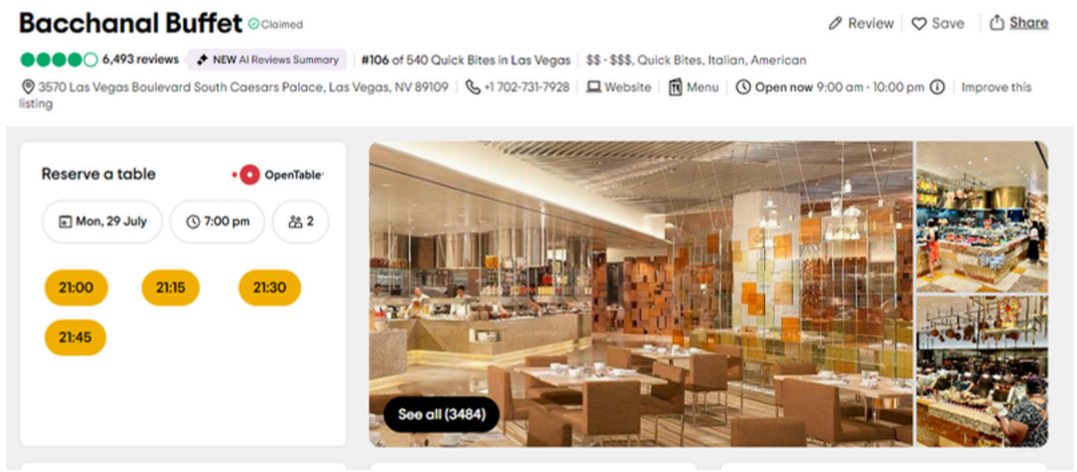
Objectives

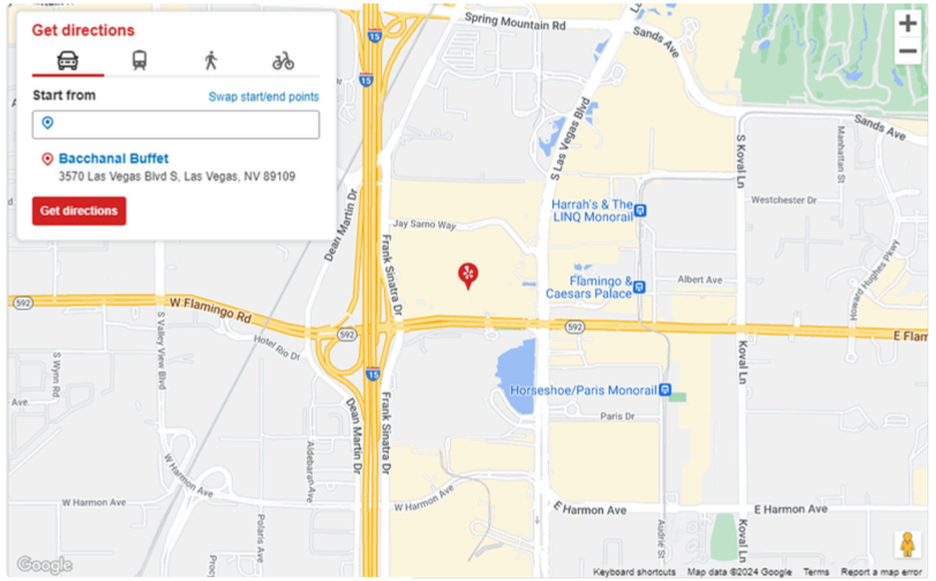
Analyze Customer Sentiments: Overall sentiment of customer reviews and identify key areas of customer satisfaction and dissatisfaction.

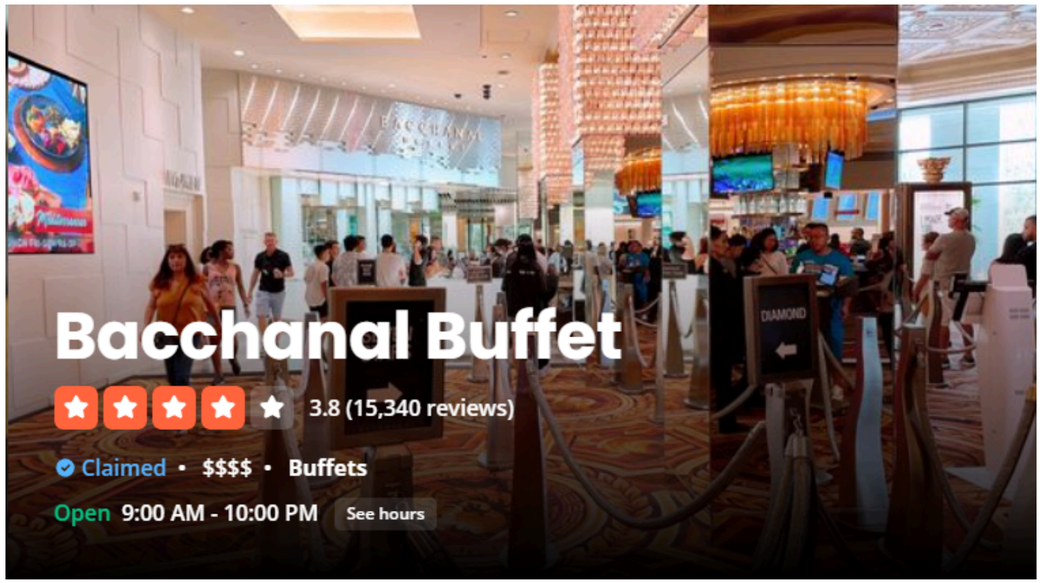
Visualize Data: Create clear and informative visualizations to present sentiment trends and patterns

Provide Actionable Insights: Offer recommendations based on sentiment analysis

Introduction



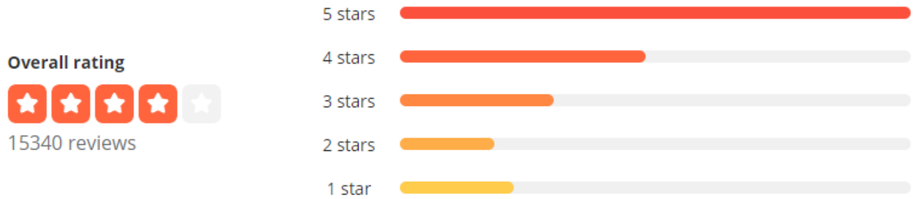




THE LARGEST BUFFET IN LAS VEGAS

Over 25,000 square feet to indulge on flavors from around the globe.

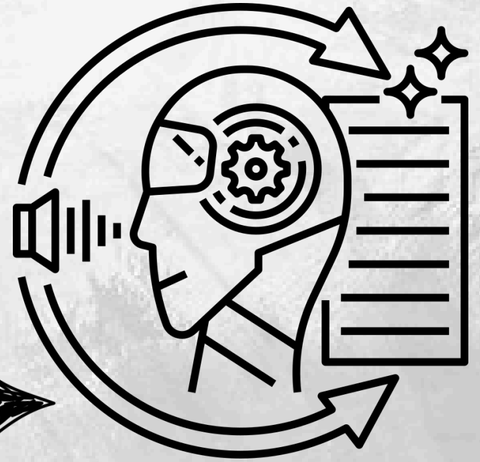
10	9	250+	∞
KITCHENS	CHEF-ATTENDED ACTION STATIONS	MENU ITEMS	INFINITE FLAVORS



Methodology

Data Acquire

Exploratory
Data Analysis



Text Processing

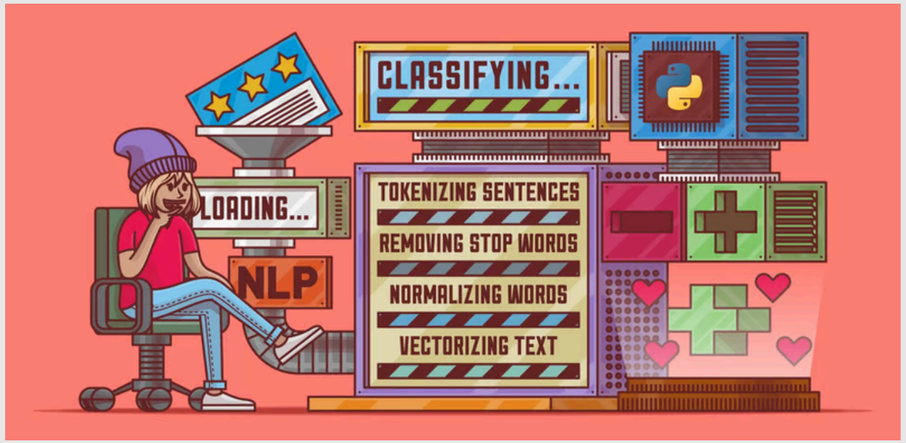
Modelling

Evaluation

Methodology

Data Acquire

Methodology



Matplotlib
Data visualisations

Pandas
Data analysis and manipulation

NumPy
Mathematical functions

jupyter

SeaBorn
Data visualisations

TextBlob

Visual Studio Code

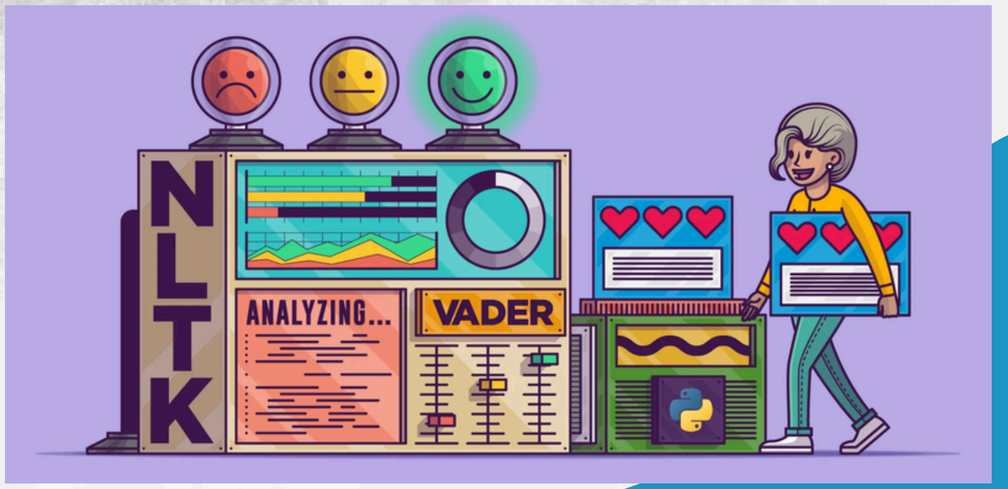
python™

wordcloud

NLTK

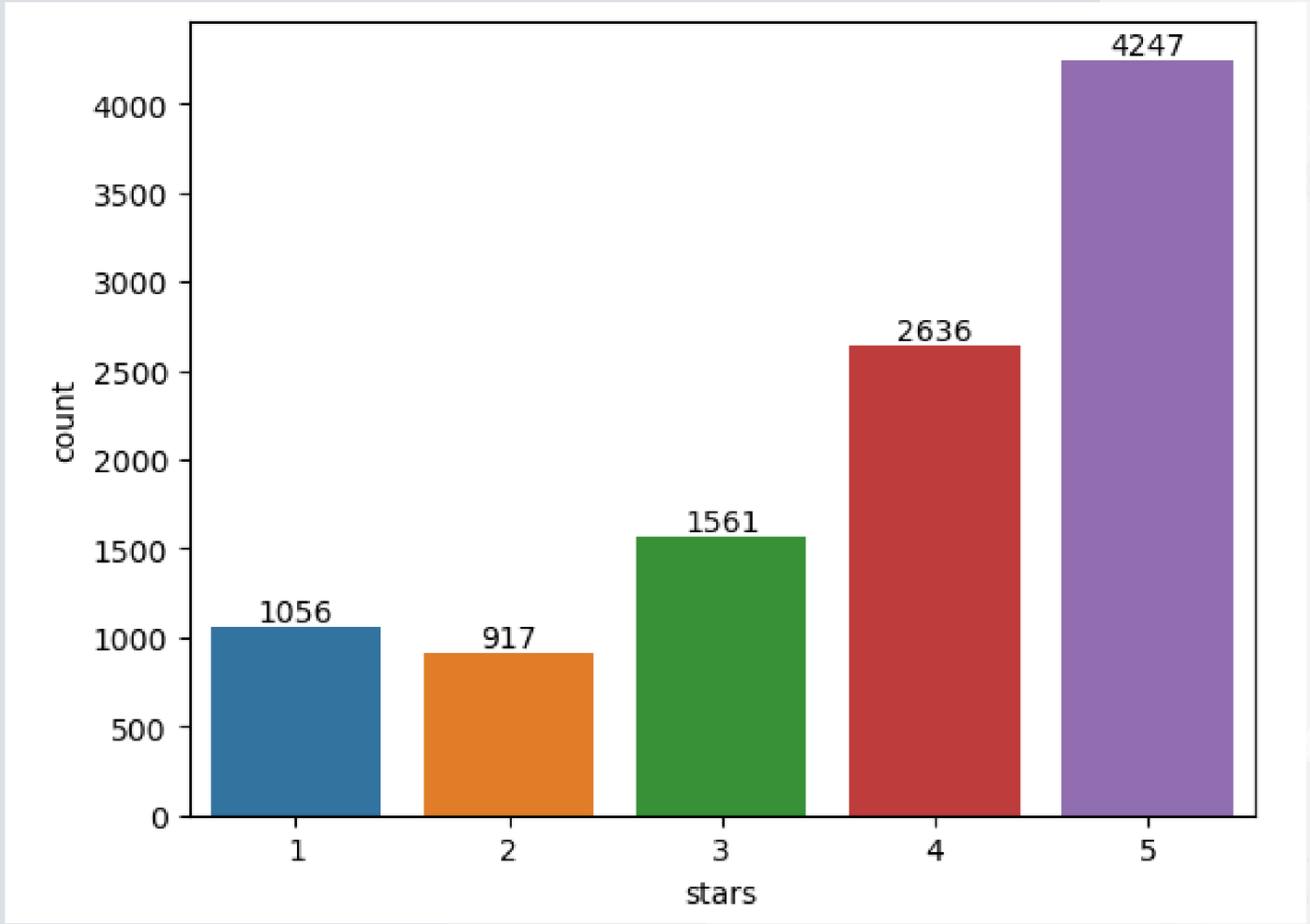
scikit learn

TOOLS



Results

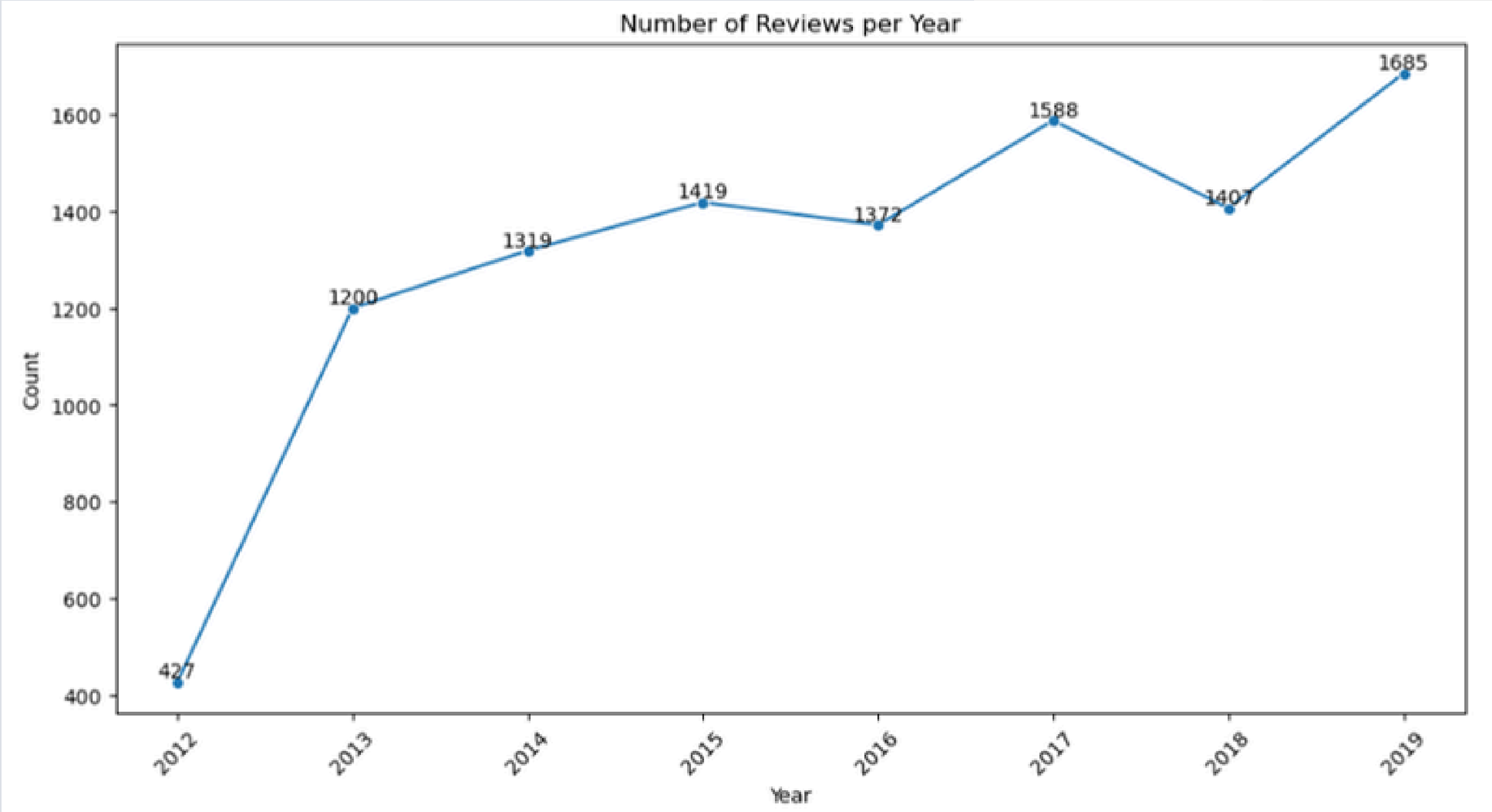
After dropping unnecessary columns



Exploratory Data Analysis

Stars distribution before text processing

Results

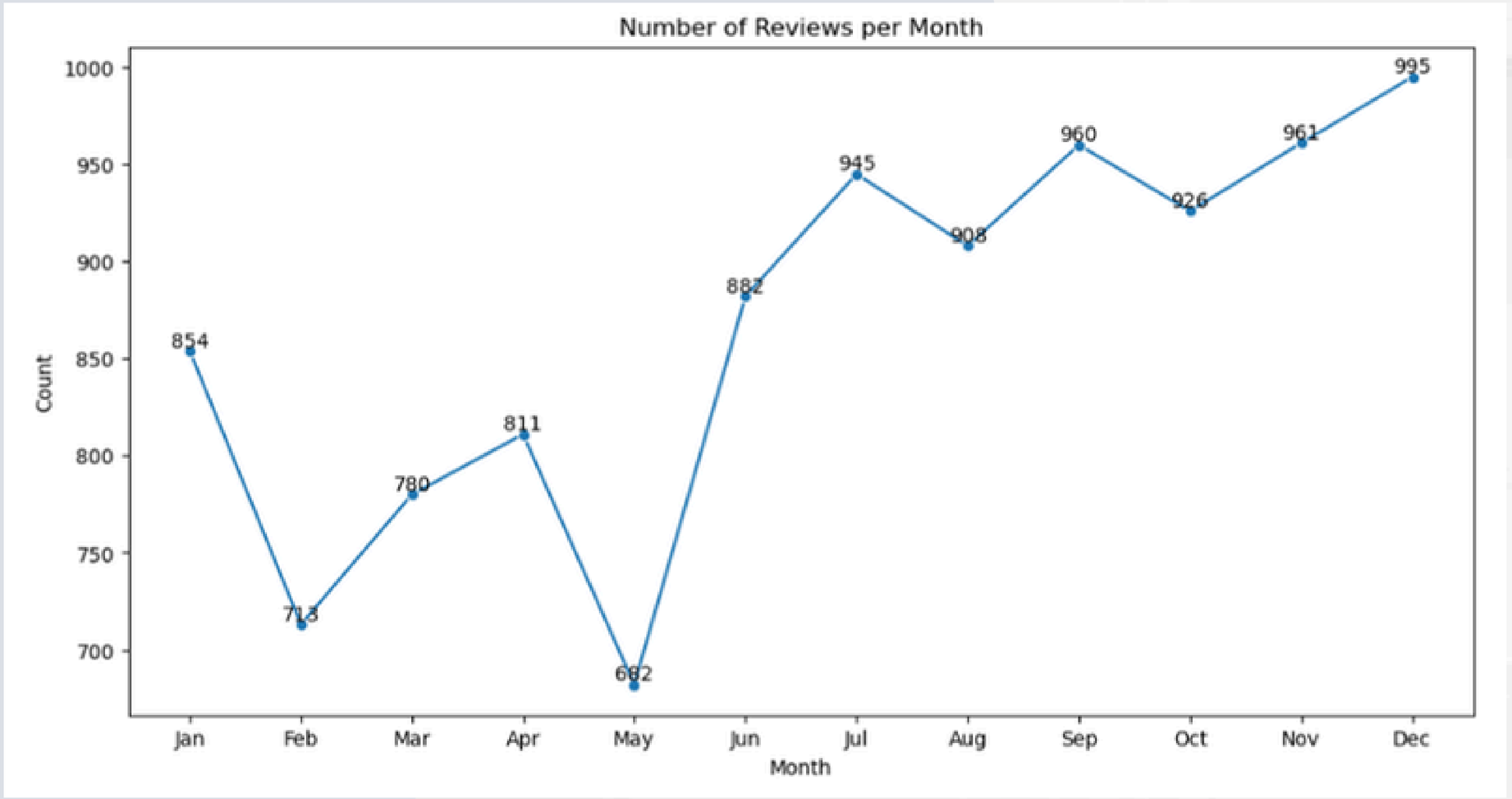


Exploratory Data Analysis

Number of Reviews per Year

Results

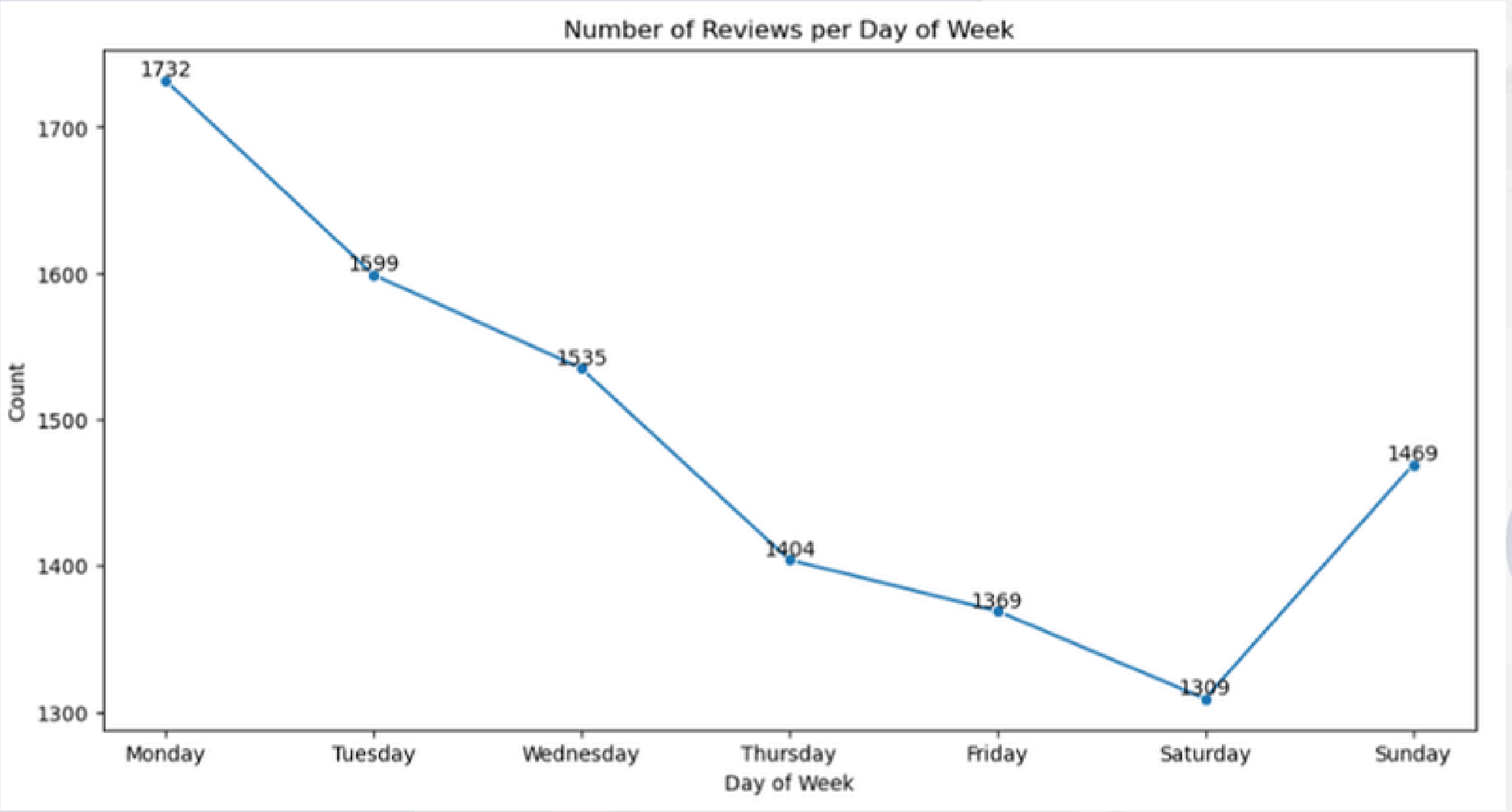
Exploratory Data Analysis



Number of Reviews per Month

Results

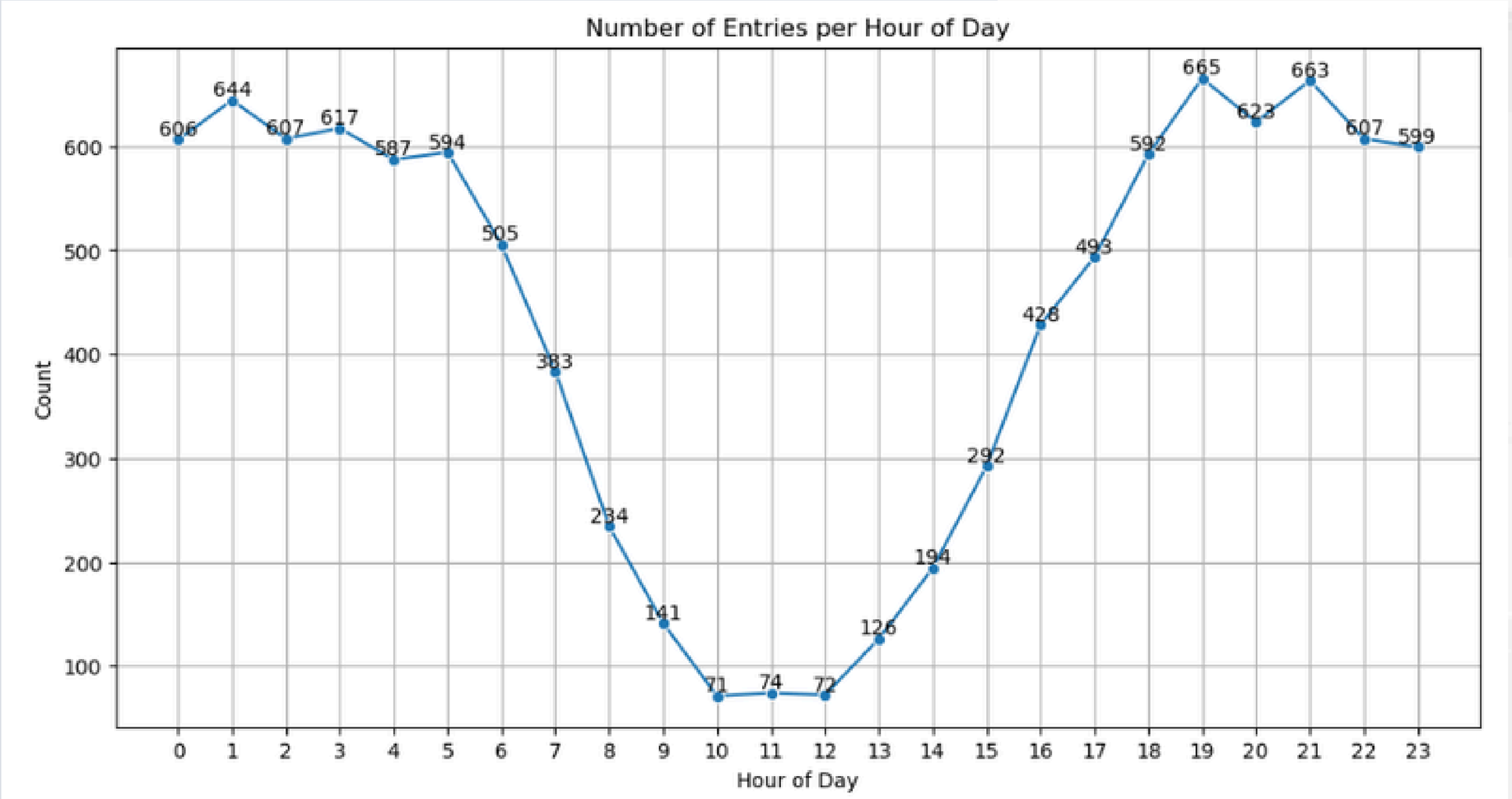
Exploratory Data Analysis



Number of Reviews per DayofWeek

Results

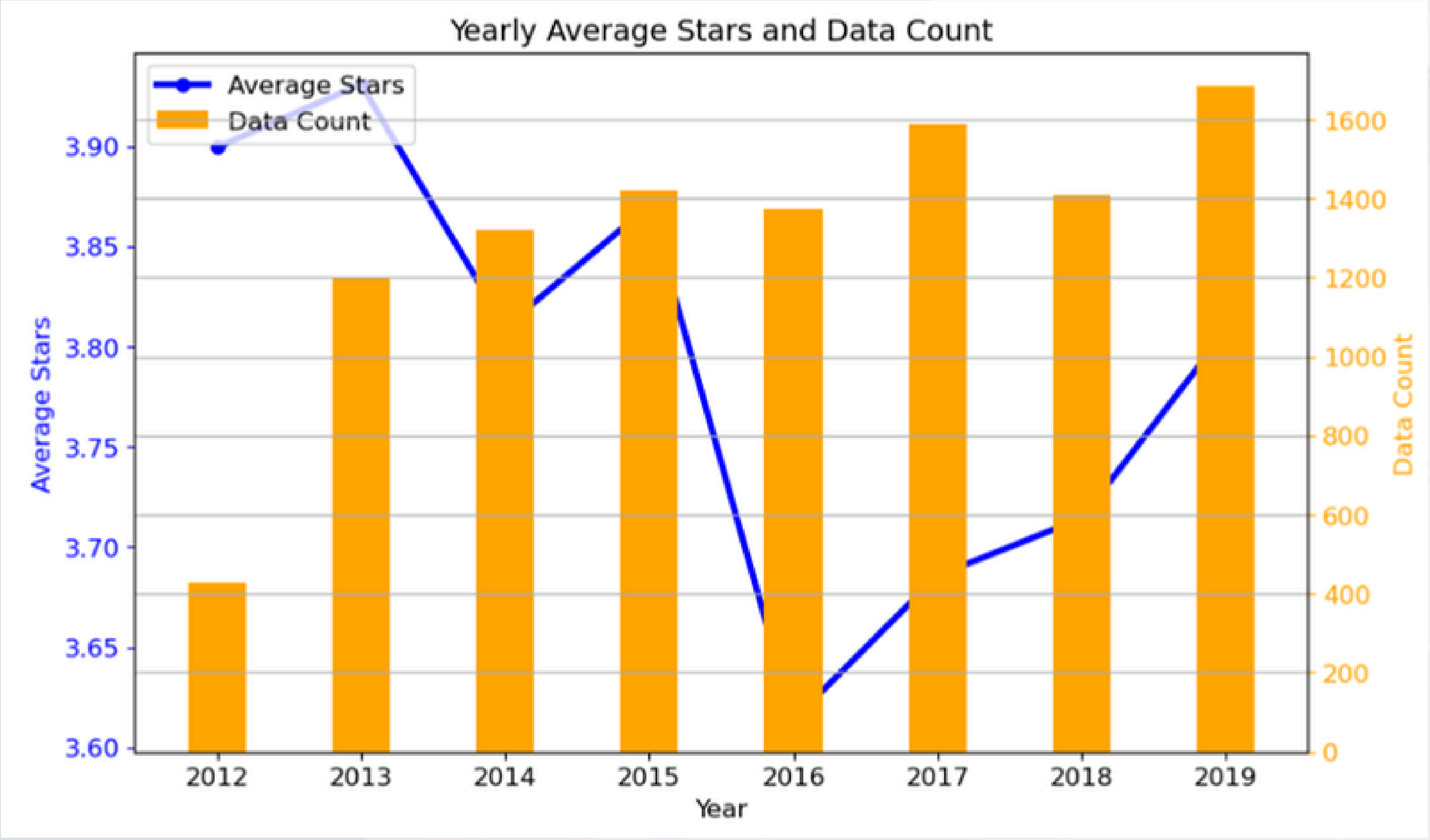
Exploratory Data Analysis



Number of Reviews along a Day

Results

Exploratory Data Analysis



Yearly avg. stars and Data Count

Results

Text Processing

Removing Characters

numbers (1-9)
punctuation ()
new line (\n)

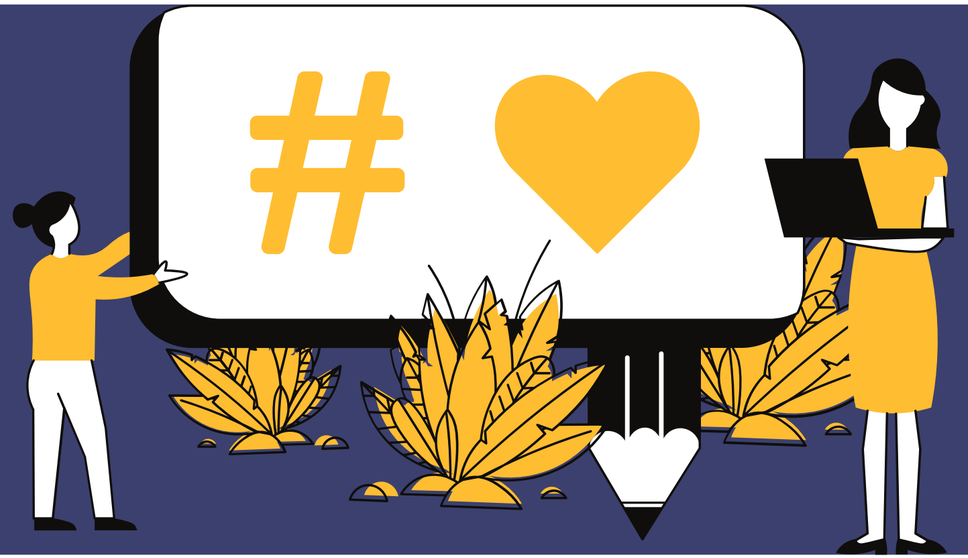
Adding Sentiment

1,2 –> Negative
3 –> Neutral
4,5 –> Positive

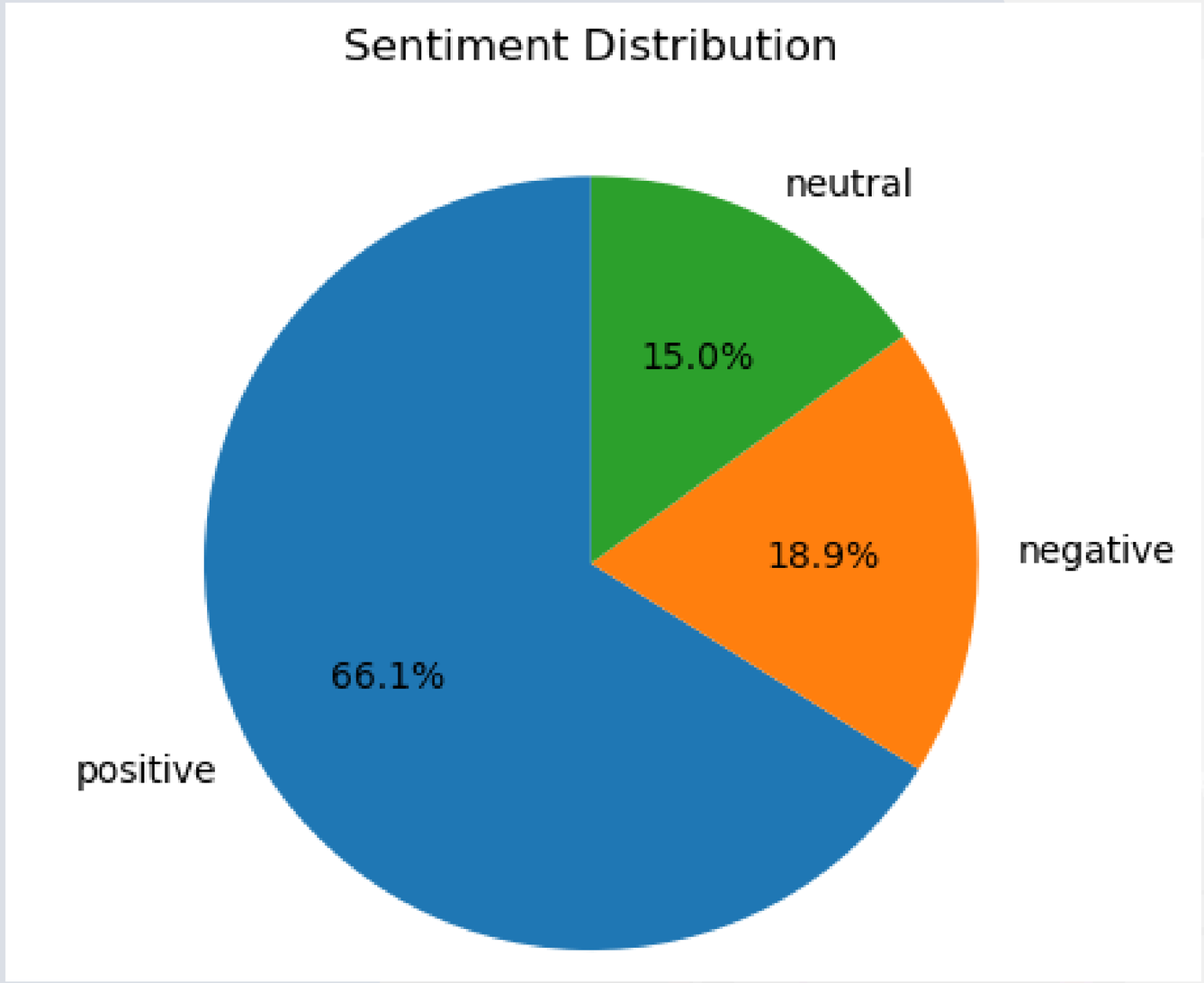
Stemming & Lemmatisation

Detecting Language

Convert Lower Case



Results

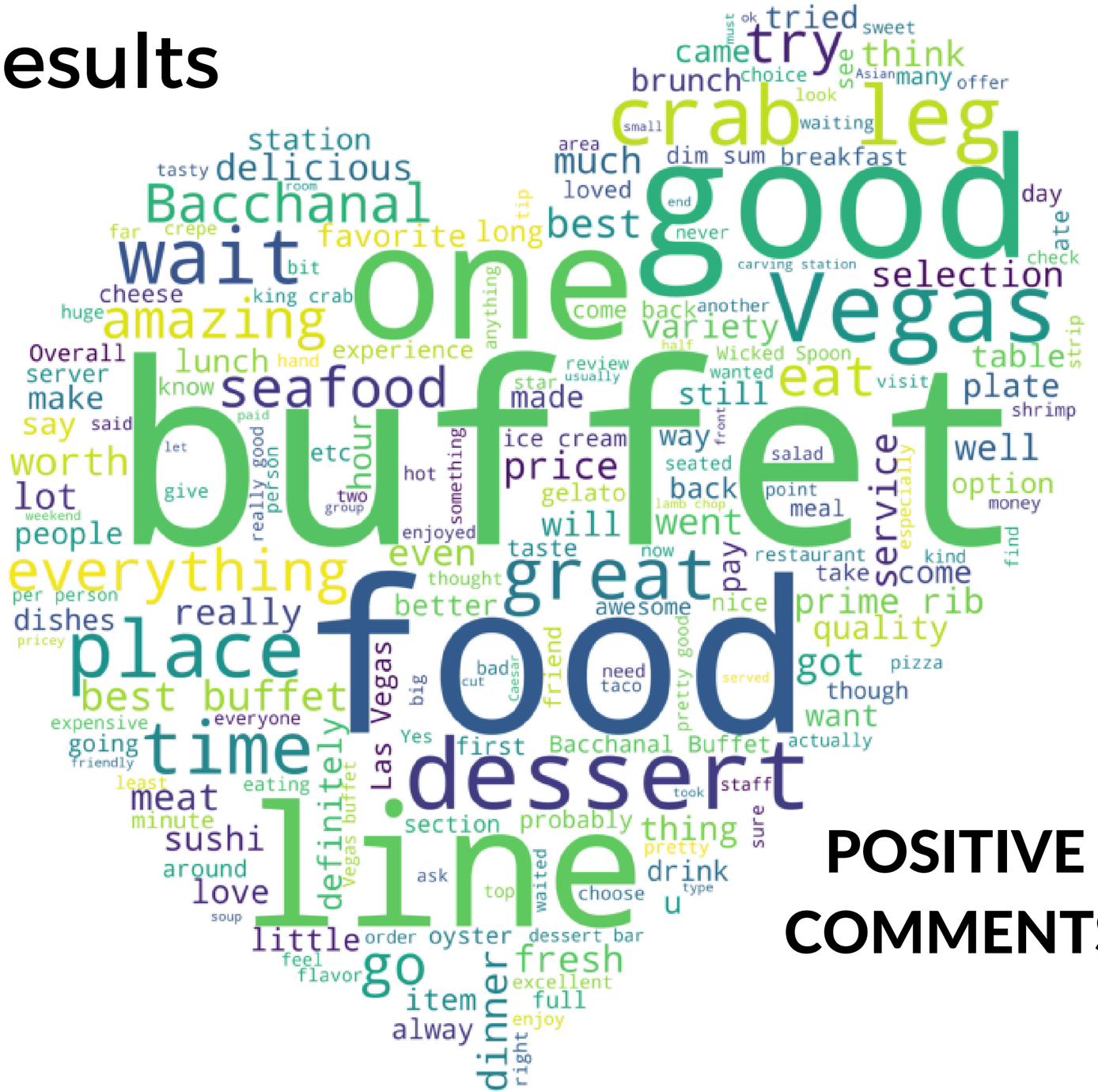


Sentiment distribution after text processing

Results

5-STAR COMMENTS

POSITIVE COMMENTS



Results

1-STAR COMMENTS

NEGATIVE COMMENTS

Results

Methods

CountVectorizer with unigram
CountVectorizer with unigram and lemmafn
CountVectorizer with bigram
CountVectorizer with bigram and lemmafn
TfidVectorizer with unigram
TfidVectorizer with unigram and lemmafn
TfidVectorizer with bigram
TfidVectorizer with bigram and lemmafn

Algorithms

LogisticRegression(),
KNeighborsClassifier(),
DecisionTreeClassifier(),
RandomForestClassifier(),
AdaBoostClassifier(),
GradientBoostingClassifier(),
MultinomialNB(),
BernoulliNB()

Results

CountVectorizes with/out ngrams or lemma

```
# 1st method: CountVectorizer with unigram
find_best_model(x, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8072289156626506,
  'f1_score': 0.8004543680700585,
  'confusion_matrix': array([[ 298,   86,   30],
         [  74,  128,   60],
         [  43,  107, 1249]], dtype=int64),
  'classification_report': '              precision    recall  f1-sc
0.72      0.72      0.72       414\n     neutral       0.40      0.4
0.93      0.89      0.91      1399\n\n    accuracy
g       0.68      0.70      0.69      2075\nweighted avg       0.82
```

```
# 2nd method: CountVectorizer with unigram and lemmafn
find_best_model(xs, y)
```

```
('Logistic Regression',
 {'accuracy': 0.7850602409638554,
  'f1_score': 0.7764937243114267,
  'confusion_matrix': array([[ 278,   81,   29],
         [  82,  111,   70],
         [  55,  129, 1240]], dtype=int64),
  'classification_report': '              precision    recall  f1-sc
0.67      0.72      0.69       388\n     neutral       0.35      0.4
0.93      0.87      0.90      1424\n\n    accuracy
g       0.65      0.67      0.66      2075\nweighted avg       0.80
```

```
# 3rd method: CountVectorizer with bigram
find_best_model(x, y)
```

```
('Logistic Regression',
 {'accuracy': 0.7966265060240963,
  'f1_score': 0.788418047812878,
  'confusion_matrix': array([[ 285,   80,   24],
         [  75,  120,   67],
         [  55,  121, 1248]], dtype=int64),
  'classification_report': '              precision    recall  f1-scc
0.69      0.73      0.71       389\n     neutral       0.37      0.46
0.93      0.88      0.90      1424\n\n    accuracy
g       0.66      0.69      0.67      2075\nweighted avg       0.82
```

```
# 4th method: CountVectorizer with bigram and lemmafn
find_best_model(xs, y)
```

```
('Logistic Regression',
 {'accuracy': 0.792289156626506,
  'f1_score': 0.7834579175763278,
  'confusion_matrix': array([[ 281,   81,   27],
         [  80,  115,   64],
         [  54,  125, 1248]], dtype=int64),
  'classification_report': '              precision    recall  f1-scc
0.68      0.72      0.70       389\n     neutral       0.36      0.44
0.93      0.87      0.90      1427\n\n    accuracy
g       0.66      0.68      0.67      2075\nweighted avg       0.81
```

Results

TfidVectorizes with/out ngrams or lemma

```
# 5th method: TfidVectorizer with unigram
find_best_model(x, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8144578313253013,
  'f1_score': 0.7935681003686282,
  'confusion_matrix': array([[ 295,   68,   16],
         [  41,   96,   24],
         [  79,  157, 1299]], dtype=int64),
  'classification_report': '              precision    recall
0.71      0.78      0.74       379\n     neutral       0.30
0.97      0.85      0.90      1535\n\n    accuracy
g       0.66      0.74      0.68      2075\nweighted avg
```

```
# 6th method: TfidVectorizer with unigram and lemmafn
find_best_model(xs, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8028915662650602,
  'f1_score': 0.7787191133218953,
  'confusion_matrix': array([[ 287,   70,   17],
         [  49,   81,   24],
         [  79,  170, 1298]], dtype=int64),
  'classification_report': '              precision    recall
0.69      0.77      0.73       374\n     neutral       0.25
0.97      0.84      0.90      1547\n\n    accuracy
g       0.64      0.71      0.66      2075\nweighted avg
```

```
# 7th method: TfidVectorizer with bigram
find_best_model(x, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8004819277108434,
  'f1_score': 0.7730826063237212,
  'confusion_matrix': array([[ 274,   64,   10],
         [  48,   76,   18],
         [  93,  181, 1311]], dtype=int64),
  'classification_report': '              precision    reca
0.66      0.79      0.72       348\n     neutral       0.24
0.98      0.83      0.90      1585\n\n    accuracy
g       0.63      0.72      0.65      2075\nweighted avg
```

```
# 8th method: TfidVectorizer with bigram and lemmafn
find_best_model(xs, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8004819277108434,
  'f1_score': 0.7732209243189241,
  'confusion_matrix': array([[ 285,   66,   16],
         [  51,   72,   19],
         [  79,  183, 1304]], dtype=int64),
  'classification_report': '              precision    reca
0.69      0.78      0.73       367\n     neutral       0.22
0.97      0.83      0.90      1566\n\n    accuracy
g       0.63      0.71      0.65      2075\nweighted avg
```

Results

```
# 5th method: TfidVectorizer with unigram
find_best_model(x, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8144578313253013,
  'f1_score': 0.7935681003686282,
  'confusion_matrix': array([[ 295,   68,   16],
         [  41,   96,   24],
         [  79,  157, 1299]], dtype=int64),
  'classification_report': '              precision    recall
0.71      0.78      0.74       379\n     neutral       0.30
0.97      0.85      0.90      1535\n\n    accuracy
g       0.66      0.74      0.68      2075\nweighted avg
```

```
# 6th method: TfidVectorizer with unigram and lemmafn
find_best_model(xs, y)
```

```
('Logistic Regression',
 {'accuracy': 0.8028915662650602,
  'f1_score': 0.7787191133218953,
  'confusion_matrix': array([[ 287,   70,   17],
         [  49,   81,   24],
         [  79,  170, 1298]], dtype=int64),
  'classification_report': '              precision    recall
0.69      0.77      0.73       374\n     neutral       0.25
0.97      0.84      0.90      1547\n\n    accuracy
g       0.64      0.71      0.66      2075\nweighted avg
```

As a result, TfidVect. with unigram provides best accuracy

Conclusion

POSITIVES

UNIGRAMS

WORD	COUNTS
buffet	391.910317
food	342.609191
good	261.634207
vegas	247.394345
best	238.342973
great	221.828242
worth	194.307755
wait	194.115807
crab	192.557076
time	187.840749

BIGRAMS

WORD	COUNTS
crab legs	86.314311
best buffet	85.819733
buffet vegas	56.919096
las vegas	47.284310
prime rib	46.548630
bacchanal buffet	31.381918
wicked spoon	30.290632
quality food	29.182357
dim sum	29.101293
king crab	27.711923

TRIGRAMS

WORD	COUNTS
best buffet vegas	34.635155
buffet las vegas	21.661086
best buffet ve	18.535559
king crab legs	17.521715
best buffet las	15.893549
hands best buffet	11.113048
favorite buffet vegas	10.383306
best buffet strip	10.248119
far best buffet	9.493477
snow crab legs	9.266184

Conclusion

NEGATIVES

UNIGRAMS

WORD	COUNTS
food	107.979998
buffet	98.069376
line	74.342726
good	68.722153
wait	67.332366
time	58.486110
crab	57.632429
just	54.318752
worth	51.350148
place	51.292444

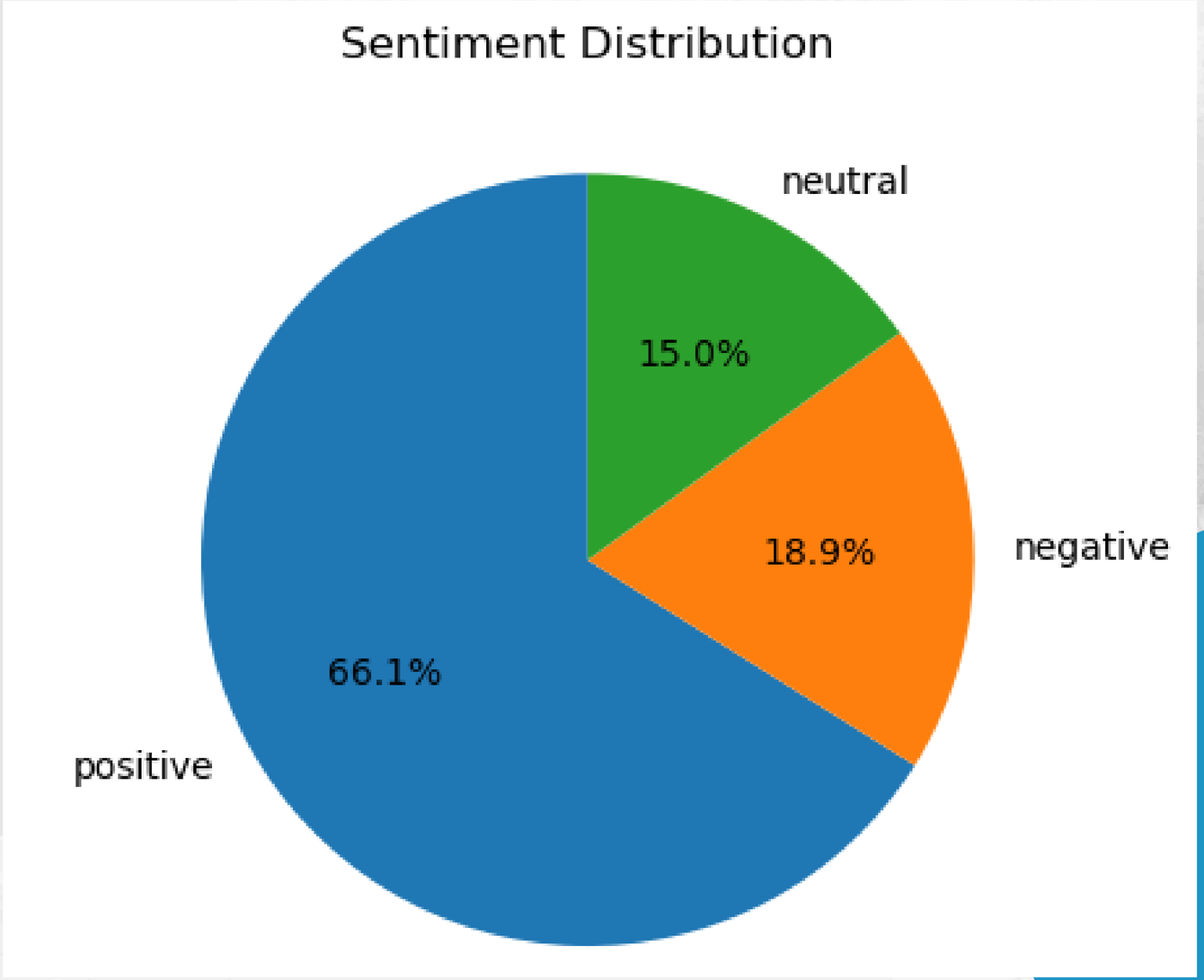
BIGRAMS

WORD	COUNTS
crab legs	22.400406
prime rib	13.396332
king crab	12.204996
wicked spoon	10.685458
food good	9.708715
quality food	9.376912
wait line	9.035394
las vegas	8.776860
buffet vegas	8.705305
bacchanal buffet	8.067720

TRIGRAMS

WORD	COUNTS
king crab legs	5.302003
best buffet vegas	3.865627
snow crab legs	3.303073
don waste money	2.589910
buffet las vegas	2.587586
got food poisoning	2.440797
definitely worth price	2.202085
waste time money	1.977260
wicked spoon better	1.966614
don waste time	1.850055

Conclusion


Sentiment Distribution
neutral
15.0%
18.9%
negative
66.1%
positive

Thanks

0530 308 63 34 afizgordu@gmail.com