GROUP 9

TEXT MINING AND PREDICTIVE ANALYSIS

ANVITA RAYALA **RAMI HUU NGUYEN** **ONKAR KHEDKAR**

Date: 04/29/2024

TABLE OF CONTENTS

INTRODUCTION TO YELP DATASET

INTRODUCTION TO YELP DATASET

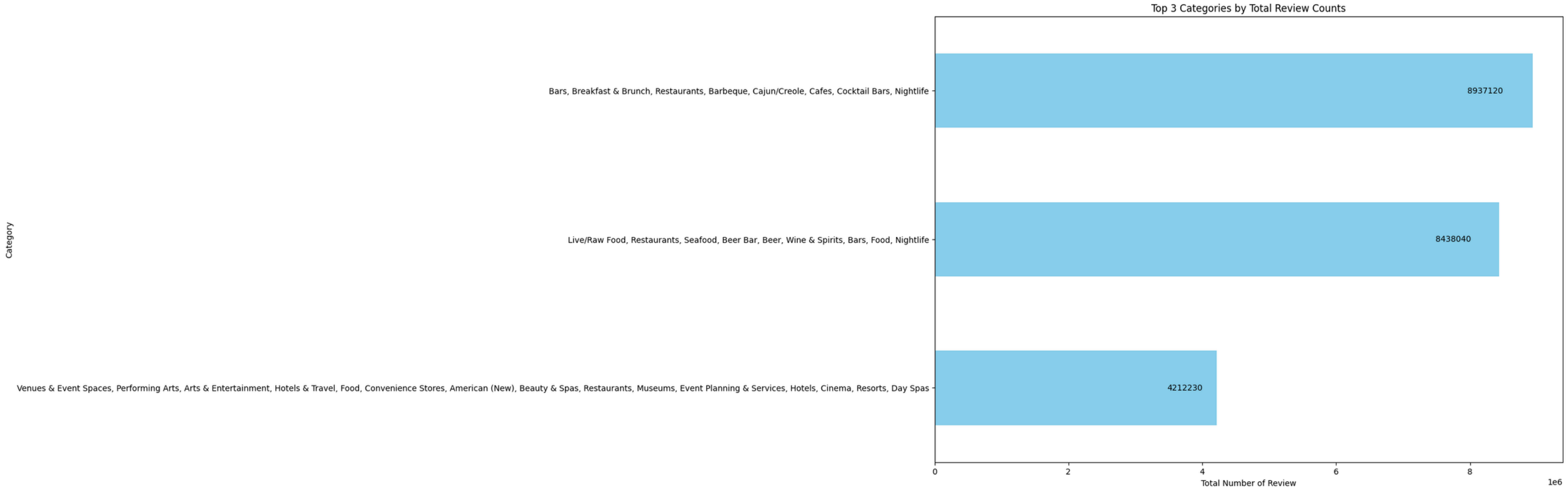
- This dataset is part of Yelp's big collection of information about **businesses, reviews, and users.**
- We took **100,000 reviews** out of a larger group of **6,990,280 reviews.**
- This smaller dataset includes information about businesses in eight big cities in the USA and Canada.
- Include 21 attributes, but we mainly focus on two key attributes: **review text, and review rating for this NLP analysis.**
- Our analysis is **useful for businesses** as it understand customers' experiences and provide better services in the future.

```
yelp_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 22 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   name          100000 non-null  object 
 1   address       99180 non-null   object 
 2   city          100000 non-null  object 
 3   state         100000 non-null  object 
 4   postal_code   100000 non-null  object 
 5   latitude      100000 non-null  float64
 6   longitude     100000 non-null  float64
 7   stars_x       100000 non-null  float64
 8   review_count  100000 non-null  int64  
 9   is_open       100000 non-null  int64  
 10  attributes    98710 non-null   object 
 11  categories    100000 non-null  object 
 12  hours         94500 non-null   object 
 13  review_id     100000 non-null  object 
 14  user_id       100000 non-null  object 
 15  stars_y       100000 non-null  int64  
 16  useful        100000 non-null  int64  
 17  funny         100000 non-null  int64  
 18  cool          100000 non-null  int64  
 19  text          100000 non-null  object 
 20  date          100000 non-null  object 
 21  sentiment     100000 non-null  object 
dtypes: float64(3), int64(6), object(13)
memory usage: 16.8+ MB
```

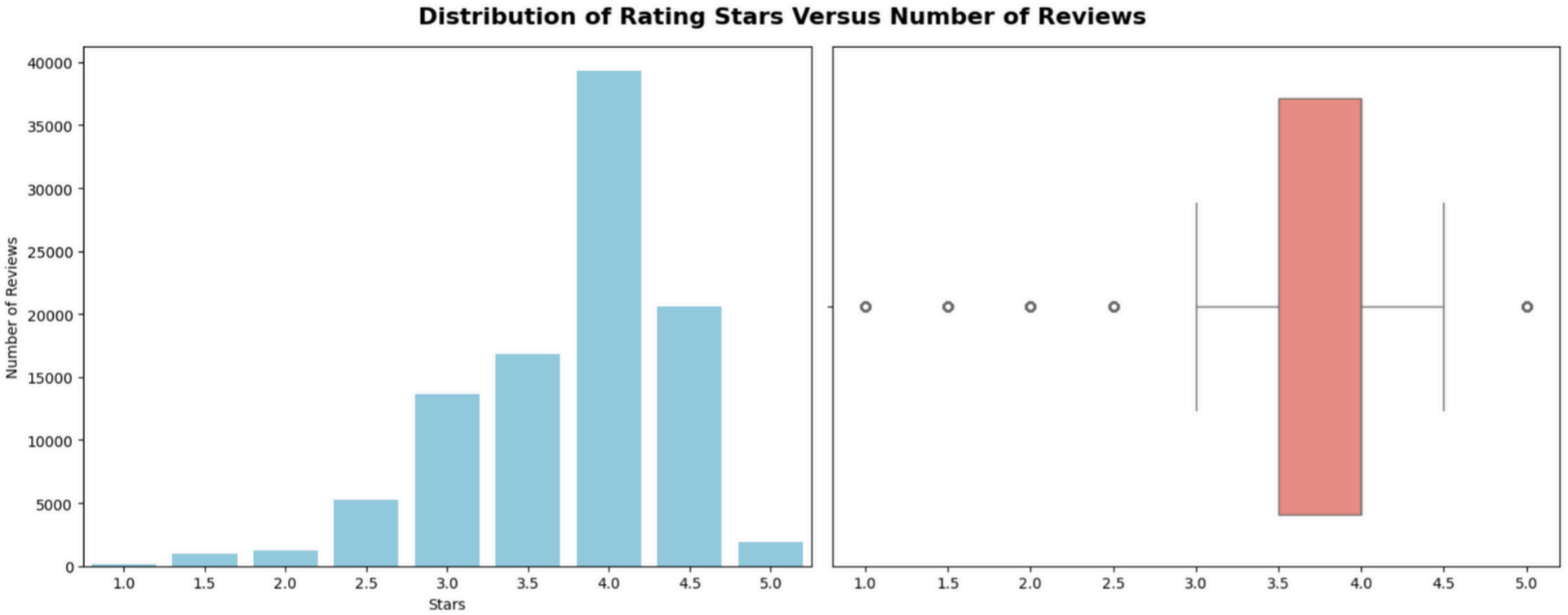
START WITH AN EXPLORATORY ANALYSIS

TOP 3 BUSINESS CATEGORIES



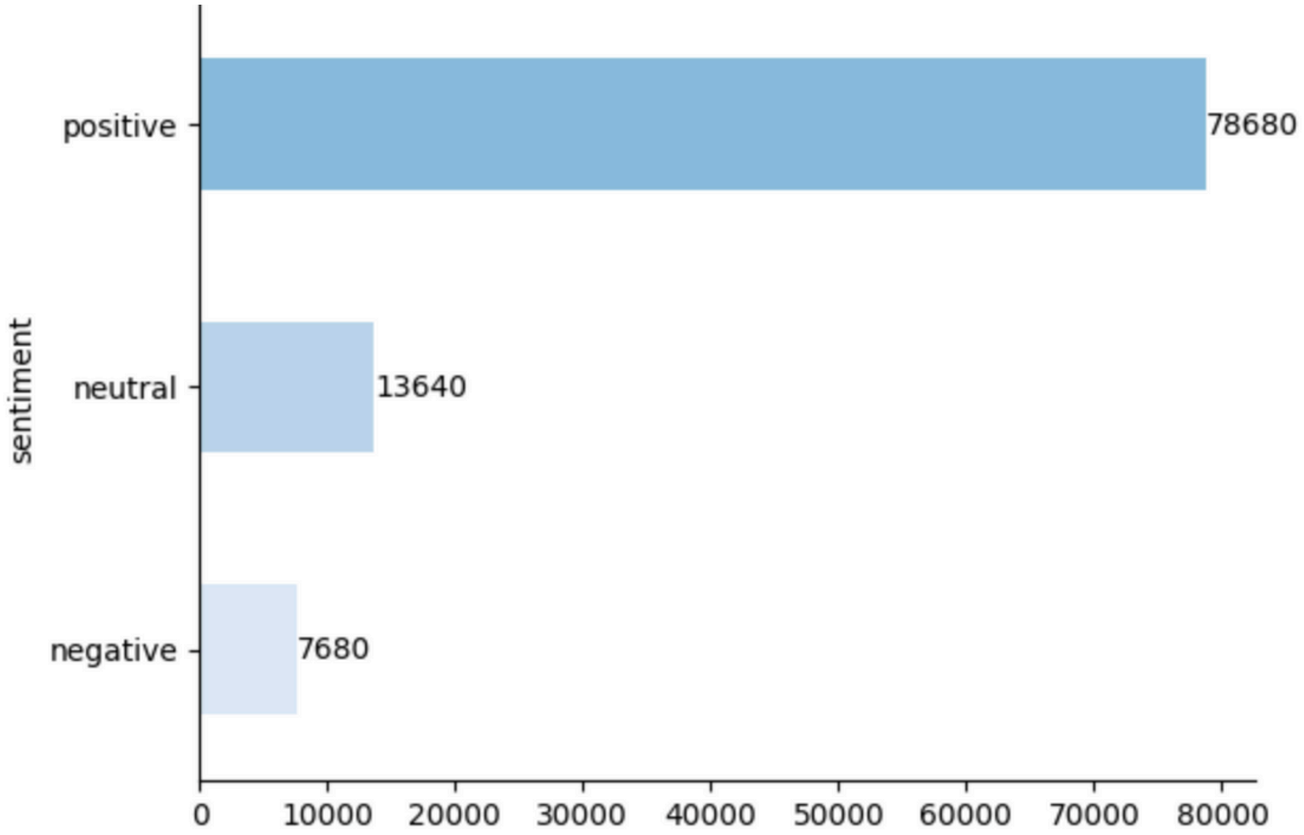
- **Dining & Nightlife** is the most reviewed category, indicating it is the most frequented and possibly the most preferred.
- **Seafood & Beverages** stands out for its focused appeal, especially among customers interested in specific food and drink options.
- **Diverse Services**, though less reviewed, caters to a wide range of interests, suggesting a niche but vital market.

DISTRIBUTION OF STARS RATING AND REVIEWS COUNT



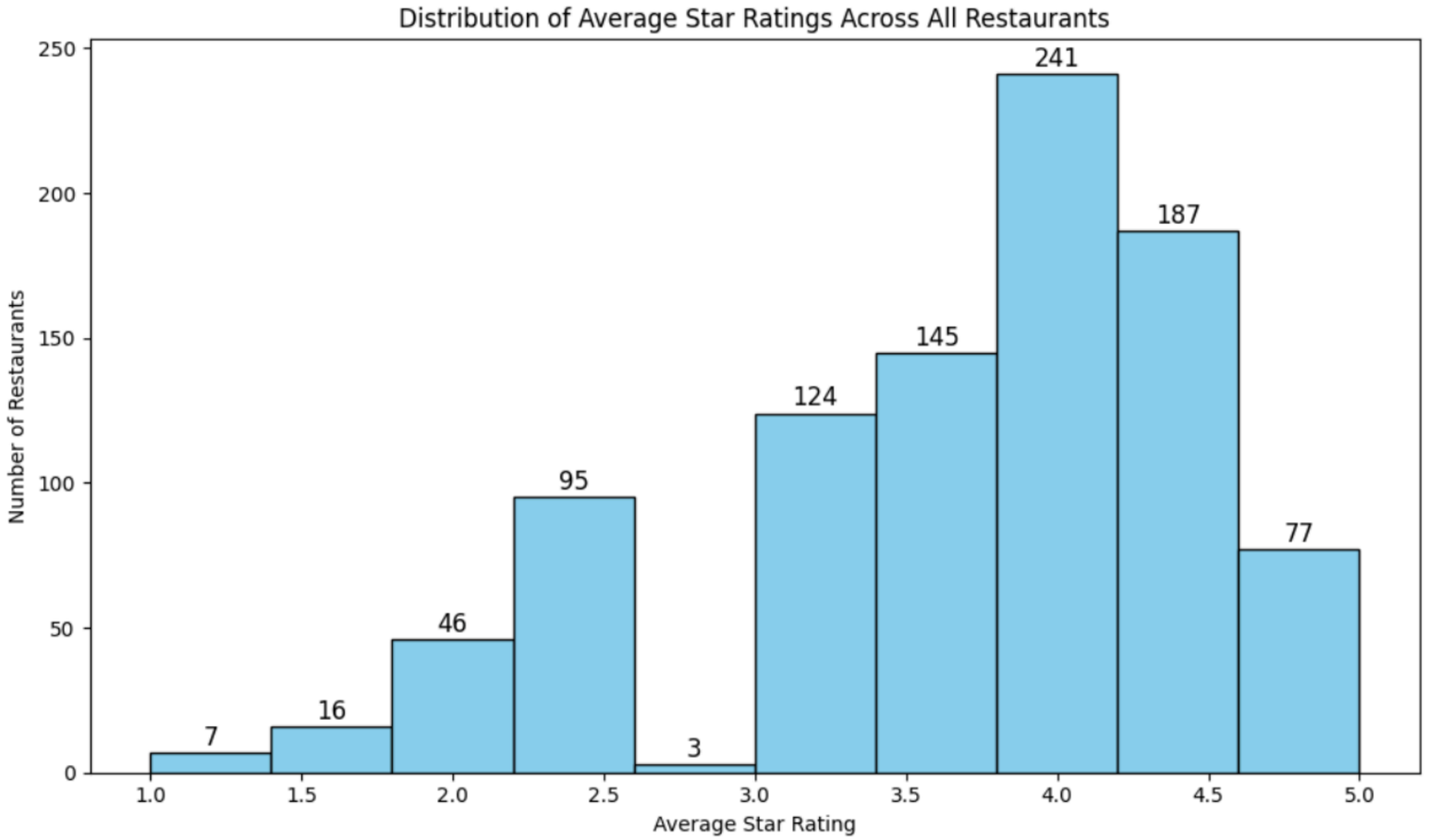
- **Positive Trend:** Majority of reviews are positively skewed, peaking at 4 stars.
- **Moderate Dominance:** 3 to 4-star ratings are most frequent, showing moderate satisfaction.
- **Extreme Outliers**:Outliers indicate rare but notable cases of very high or very low ratings.

SENTIMENT AND STAR RATING DISTRIBUTION



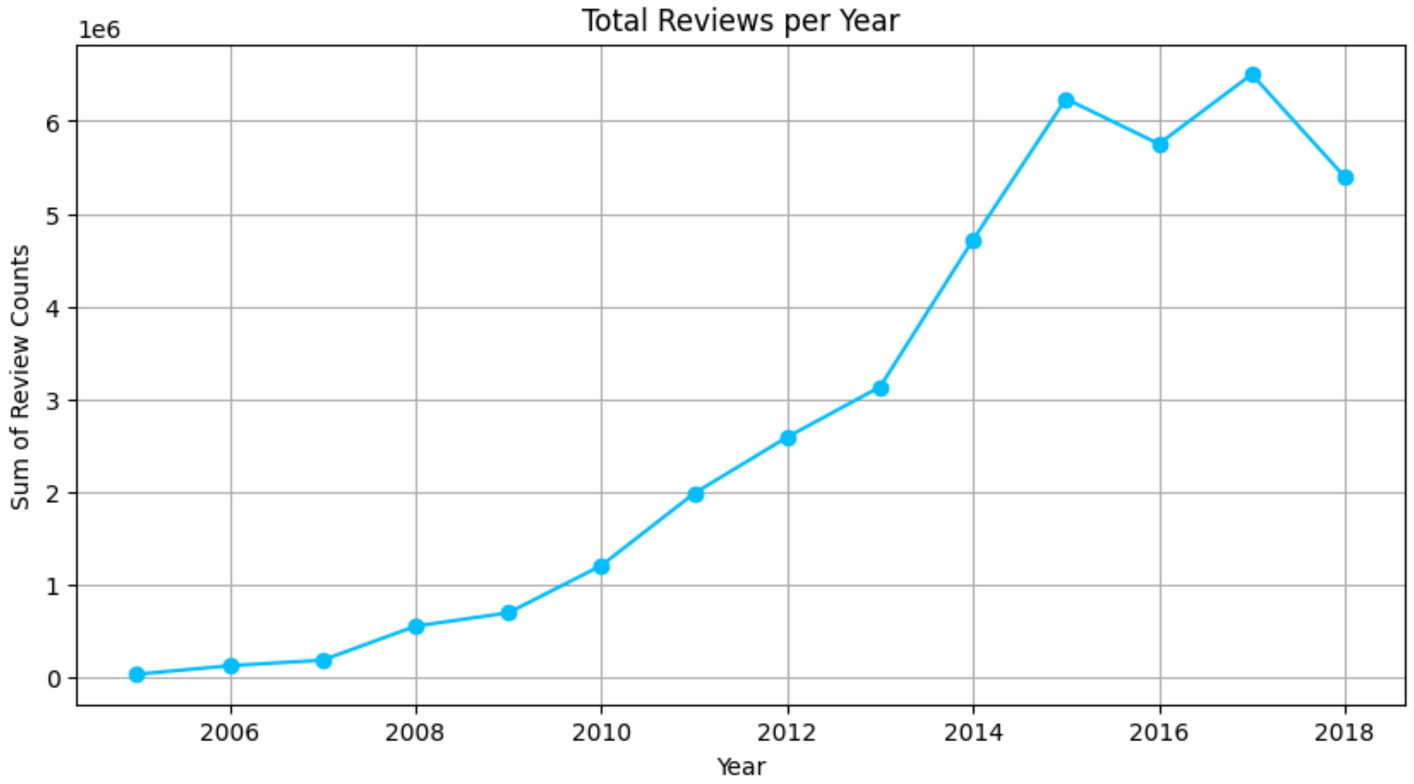
- **Dominant Positivity**: 78,680 reviews exhibit positive sentiment, where users are more inclined to share favorable experiences.
- **Moderate Neutrality:** 13,640 reviews, reflects neutral sentiment, indicating a balanced perspective on user experiences.
- **Lower Negativity:** 7,680 reviews express negative sentiment,dissatisfaction are relatively uncommon among users.

BEST RESTAURANTS VERSUS STAR RATING DISTRIBUTION



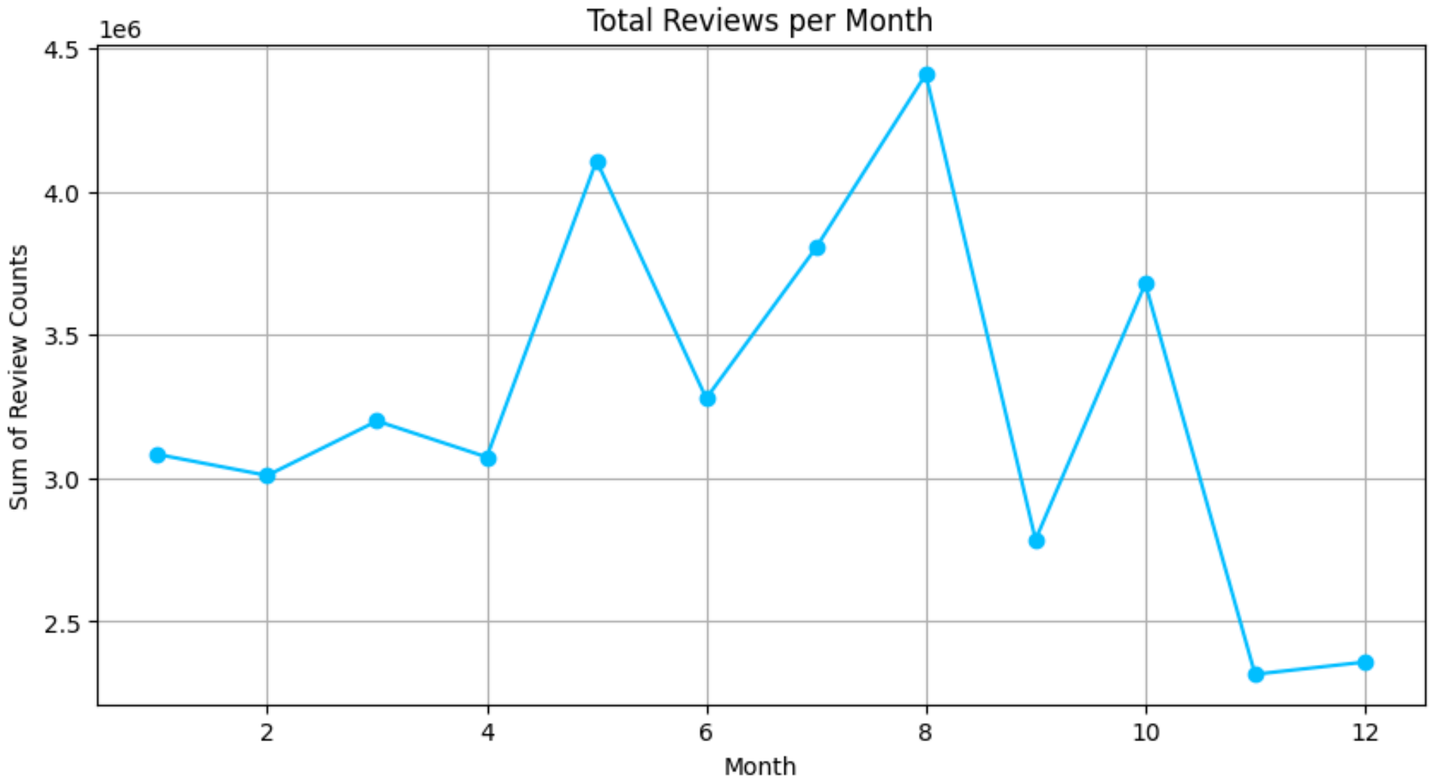
The majority of restaurants achieving a **4-star average rating** indicates a high level of overall customer satisfaction.

YEARLY REVIEWS COUNT



Illustrate **a clear upward trend** in the total number of reviews from 2006 to 2018, with a slight dip after 2016. This suggests a growing user engagement over the years on the platform.

MONTHLY REVIEWS COUNT



Peaks in review activity during the **third, fifth, and eight, and ten months,** potentially aligning with spring breaks, summer vacations, and holiday seasons.

TEXT MINING

WHAT ARE FREQUENT KEYWORDS MENTIONED IN GOOD AND BAD REVIEWS DATASET?

POS PATTERN EXTRACTION AND WORDCLOUDS

ANALYZE 100 REVIEW SAMPLES USING 9 POS TAGGING PATTERNS.

```
# Define the POS tagging patterns for tips
tip_patterns = [
    [{"POS": "ADJ"}, {"POS": "NOUN"}],  # e.g., decent variety
    [{"POS": "ADJ"}, {"POS": "NOUN"}, {"POS": "NOUN"}], # e.g., red bean filled
    [{"POS": "VERB"}, {"POS": "ADJ"}, {"POS": "NOUN"}], # e.g., try delicious food
    [{"POS": "VERB"}, {"POS": "NOUN"}, {"POS": "ADJ"}], # e.g., serve coffee hot
    [{"POS": "ADV"}, {"POS": "VERB"}, {"POS": "NOUN"}], # e.g., quickly grab tickets
    [{"POS": "NOUN"}, {"POS": "ADP"}, {"POS": "NOUN"}], # e.g., table inside restaurant
    [{"POS": "VERB"}, {"POS": "NOUN"}, {"POS": "NOUN"}], # e.g., buy dozen eggs
    [{"POS": "ADJ"}, {"POS": "NOUN"}, {"POS": "ADP"}, {"POS": "NOUN"}], # e.g., great place for kids
    [{"POS": "ADV"}, {"POS": "ADJ"}, {"POS": "NOUN"}] # e.g., absolutely perfect come
]
```

APPLY THESE PATTERNS INTO GOOD AND BAD REVIEWS DATASET

REVIEWS FROM CUSTOMERS WHO RATED THEIR EXPERIENCE AS 3 STARS OR HIGHER.

REVIEWS FROM CUSTOMERS WHO RATED THEIR EXPERIENCE BELOW 3 STARS.

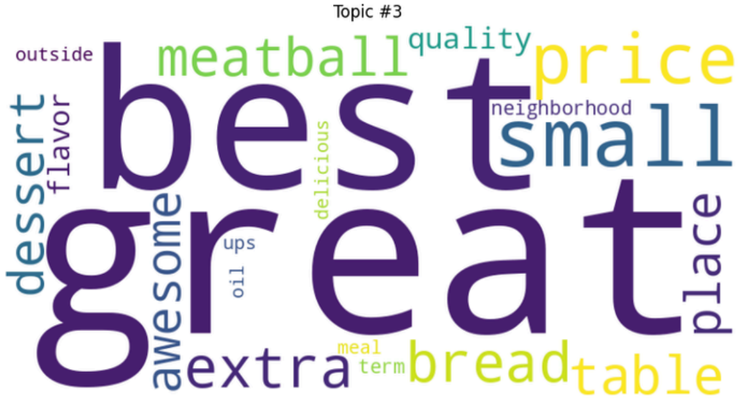
The word **"next time"** appearing in both positive and negative reviews suggests a complexity in customer experiences.
POS tagging relies on **grammatical structure**, which may not always accurately reflect sentiment or the importance of the phrase to the overall review.

TEXT MINING

WHAT ARE TOPICS MENTIONED IN GOOD REVIEW DATASET?

TOPIC MODELLING LDA

RECOGNIZING WHAT CUSTOMERS APPRECIATE THE MOST



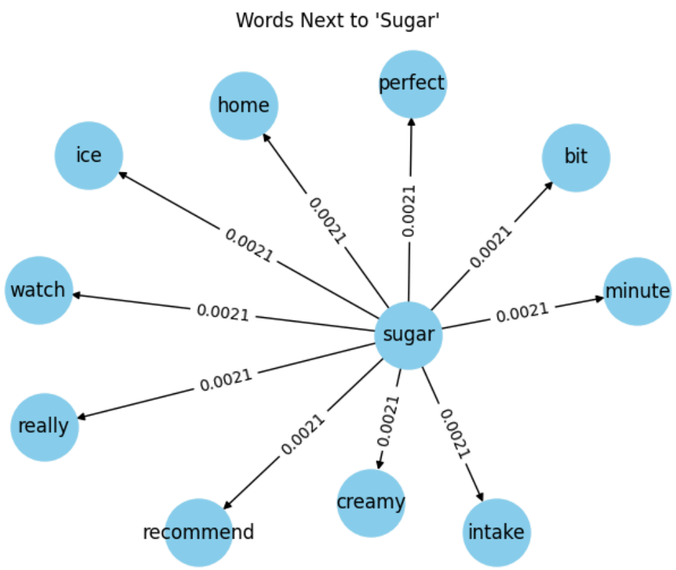
- **Authenticity and food quality** are highly valued.
- **Sweetness levels and service times** are areas that could be improved.
- **Portion size relative to price** is a significant concern for some customers.
- The word **sugar** could be positive or negative meaning.
  - Lack of insights into sentiment

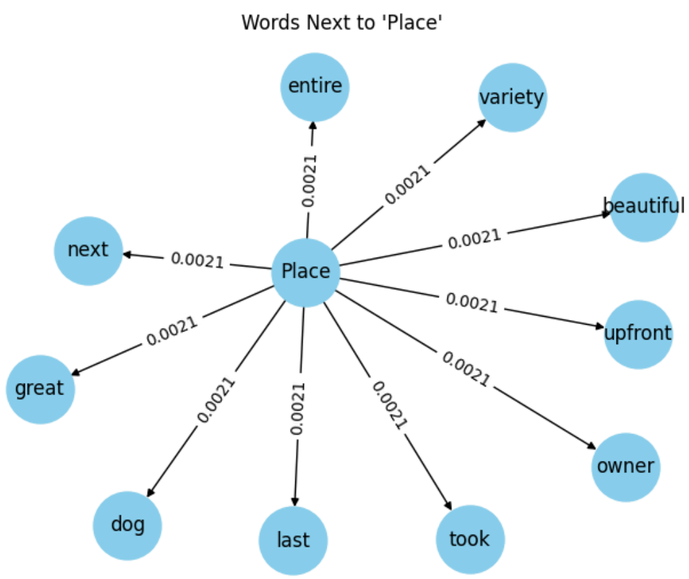
TEXT MINING

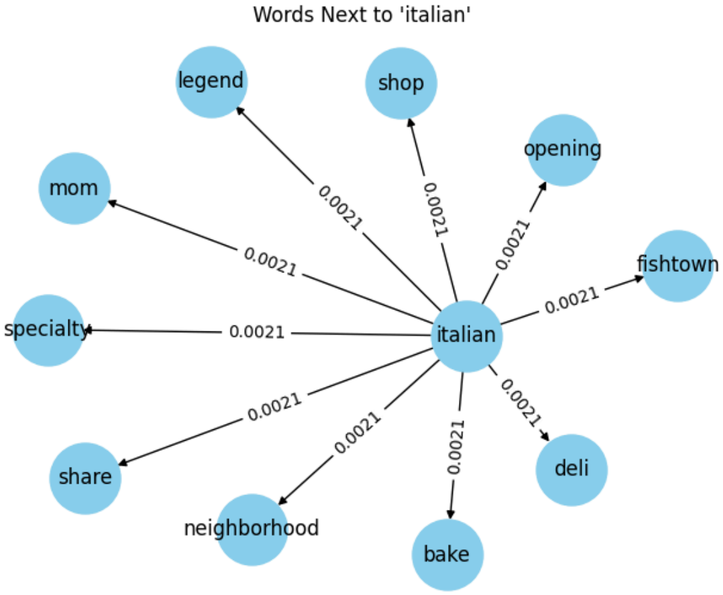
WHAT WORDS ARE NEXT TO ITALIAN, SUGAR, AND PLACE?

WORD 2 VEC

ANALYZE THE WORDS APPEAR IMMEDIATELY BEFORE AND AFTER THE TARGETED WORDS







- **Sugar** discussions often focus on its quality and quantity.
- **Positive associations with Place** suggest satisfaction with the ambiance or location.
- **Keywords surrounding Italian** highlights a strong community and family preference in its cuisines.
- **Great place** highlights to contextual ambiguity as it may refer to atmosphere or specific location?

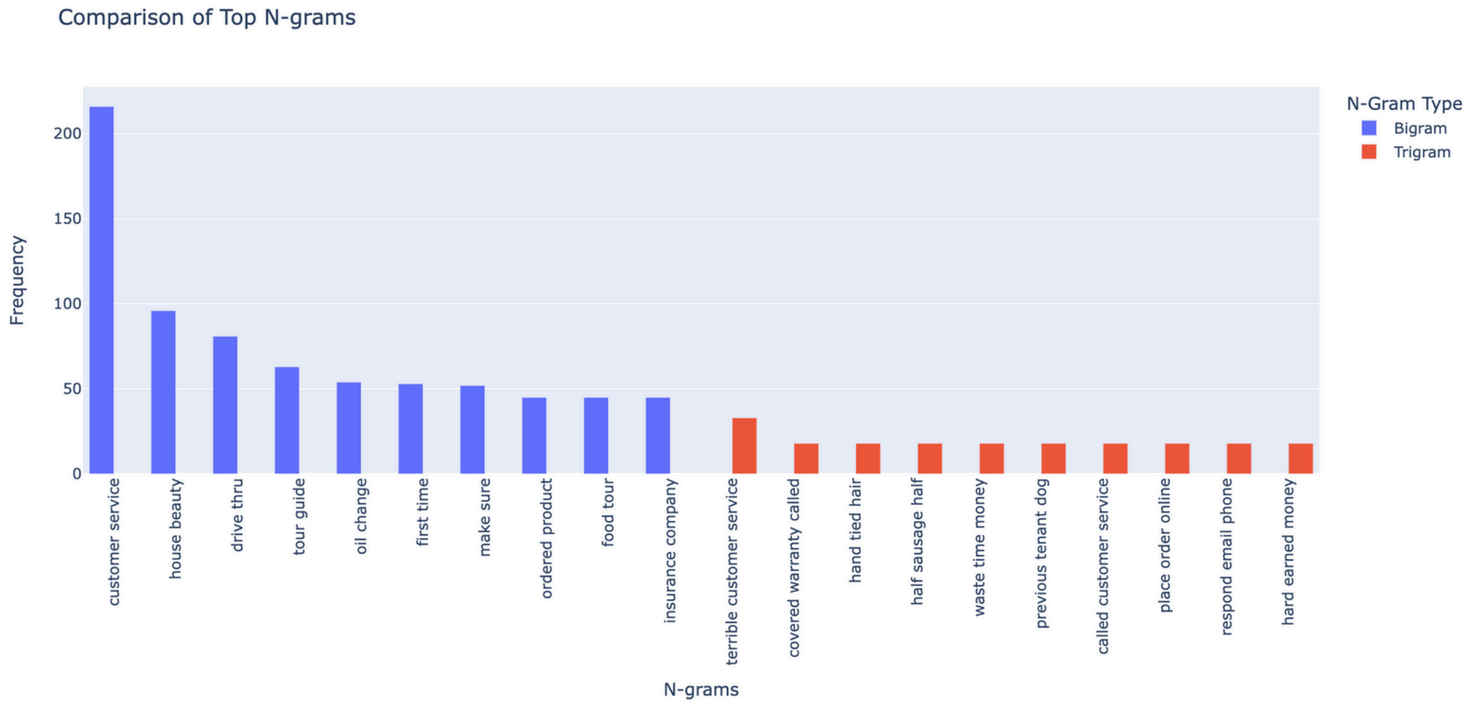
TEXT MINING

WHAT ARE THE NEGATIVE REVIEWS BEING SPOKEN BY CUSTOMERS?

N-GRAMS

A SUBSET OF 1000 SAMPLES WAS SELECTED FROM THE DATASET

REVIEWS FROM CUSTOMERS WHO RATED THEIR EXPERIENCE AS 2 STARS OR LOWER.



- Negative reviews frequently cite "**customer service**" as a problem area.
- Specific phrases like "**terrible customer service**" and "**waste time money**" suggest strong dissatisfaction with service efficiency and value.
- **Bi-gram** reflects general experiences and its aspect (customer service and drive through).
- **Tri-gram** reflects specific experience (terrible customer experience, and previous tenant dogs).
- **Lack of context** highlights in this phrase “terrible customer service” but did not state reason why it was terrible are not explained.

MOVE TO PREDICTIVE ANALYSIS

PREDICT SENTIMENTS OF YELP REVIEWS BY APPLYING 7 MODELS

1. Bag of Words with Multinomial Naive Bayes
2. TF-IDF with Multinomial Naive Bayes
3. Pretrained Naive Bayes
4. LSTM
5. RNN
6. BERT
7. RoBERTa

- **Positive sentiments for those who rated over than 3.**
- **Negative sentiments for those who rated below 3.**
- **Neutral sentiments for those who rated equal 3.**
- **Choose the best model to predict sentiment based on the accuracy level.**

BAG-OF-WORDS WITH MULTINOMIALNB

1. Convert text reviews into numerical data
2. Classify review into three categories
3. Check accuracy level of the model

```
# Importing libraries
from sklearn.feature_extraction.text import CountVectorizer

# Creating the Bag of Words model
cv = CountVectorizer(max_features=2500, binary=True, ngram_range=(1, 3))

# 'Cleaned_text' column already exists in yelp_dataset
X = cv.fit_transform(yelp_dataset['cleaned_text']).toarray()

# Mapping sentiment labels
sentiment_map = {'neutral': 0, 'positive': 1, 'negative': 2}

yelp_dataset['sentiment'] = yelp_dataset['sentiment'].map(sentiment_map)

yelp_dataset['sentiment']
```

```
0        1
1        0
2        0
3        0
4        1
       ..
99995    1
99996    1
99997    2
99998    2
99999    2
Name: sentiment, Length: 100000, dtype: int64
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, yelp_dataset['senti

# Creating and training the Multinomial Naive Bayes model
sentiment_detect_model = MultinomialNB().fit(X_train, y_train)

# Making predictions on the test set
y_pred = sentiment_detect_model.predict(X_test)

# Calculating accuracy
score = accuracy_score(y_test, y_pred)
print("Accuracy:", score)
```

```
Accuracy: 0.7895
```

TFIDF WITH MULTINOMINAL NB

1. Convert text reviews into numerical data
2. Classify review into three categories
3. Check accuracy level of the model

```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.model_selection import train_test_split
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.metrics import accuracy_score

     # Define corpus and y
     corpus = yelp_dataset['cleaned_text']
     y = yelp_dataset['sentiment']

     # Creating the TF-IDF vectorizer
     tv = TfidfVectorizer(max_features=2500, ngram_range=(1,3))
     X = tv.fit_transform(corpus).toarray()

     # Splitting the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,

     # Creating and training the Multinomial Naive Bayes model
     tfdif_detect_model = MultinomialNB().fit(X_train, y_train)

     # Making predictions on the test set
     y_pred = tfdif_detect_model.predict(X_test)

     # Calculating and printing the accuracy score
     score = accuracy_score(y_test, y_pred)
     print("Accuracy:", score)

Accuracy: 0.83055
```

PRETRAINNED NAIVE BAYES (TEXT BOB)

1. Convert text reviews into numerical data
2. Classify review into three categories
3. Check accuracy level of the model

```python
from textblob import TextBlob
from sklearn.metrics import accuracy_score

# Define a function to use TextBlob to make sentiment prediction
def sentiment_textblob(text):
    blob = TextBlob(text)
    if blob.sentiment.polarity >= 0:
        return 1  # Positive sentiment
    else:
        return 0  # Negative sentiment

# Convert X_test from array to list of strings
X_test_text = [str(text) for text in X_test]

# Make predictions using TextBlob on the test set
y_pred_textblob = [sentiment_textblob(text) for text in X_test_text]

# Calculate accuracy for TextBlob predictions
accuracy_textblob = accuracy_score(y_test, y_pred_textblob)
print("Accuracy using TextBlob:", accuracy_textblob)
```

```
Accuracy using TextBlob: 0.7879
```

LONG SHORT-TERM MEMORY (LSTM)

1. Convert text reviews into numerical sequences
2. Trainning these sequences by padding them
3. Check accuracy level of the model

```
[ ]  tokenizer = Tokenizer()
     tokenizer.fit_on_texts(X)
     X_seq = tokenizer.texts_to_sequences(X)
     max_seq_length = max([len(seq) for seq in X_seq])
     X_pad = pad_sequences(X_seq, maxlen=max_seq_length)
```

```
     X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size=0.
     vocab_size = len(tokenizer.word_index) + 1
     embedding_dim = 100

     model = Sequential()
     model.add(Embedding(vocab_size, embedding_dim, input_length=max_seq_lengtl
     model.add(LSTM(32))
     model.add(Dense(1, activation='sigmoid'))

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accu
     model.fit(X_train, y_train, epochs=5, batch_size=128, validation_split=0.1
```

```
Epoch 1/5
563/563 [==============================] - 811s 1s/step - loss: -0.2043 - a
Epoch 2/5
563/563 [==============================] - 1046s 2s/step - loss: -2.1185 -
Epoch 3/5
563/563 [==============================] - 816s 1s/step - loss: -3.6037 - a
Epoch 4/5
563/563 [==============================] - 774s 1s/step - loss: -5.0983 - a
Epoch 5/5
563/563 [==============================] - 787s 1s/step - loss: -6.4695 - a
<keras.src.callbacks.History at 0x7a8e85af4c10>
```

```
[ ]  loss, accuracy = model.evaluate(X_test, y_test)
     print(f"Test Accuracy: {accuracy}")
```

```
625/625 [==============================] - 81s 130ms/step - loss: -7.3101 -
Test Accuracy: 0.8850000205993652
```

RECURRENT NEURAL NETWORK (RNN)

1. Convert text reviews into vectors
2. Trainning these sequences by padding them
3. Check accuracy level of the model

```
[ ]  vocab_size = len(tokenizer.word_index) + 1

     model = Sequential()
     model.add(Embedding(input_dim=vocab_size, output_dim=100, input_length=ma>
     model.add(SimpleRNN(units=64))  # Simple RNN layer
     model.add(Dense(units=1, activation='sigmoid'))

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
     model.fit(X_train, y_train, epochs=5, batch_size=128, validation_split=0.1
```

```
Epoch 1/5
563/563 [==============================] - 443s 785ms/step - loss: -0.7330
Epoch 2/5
563/563 [==============================] - 407s 723ms/step - loss: -0.9715
Epoch 3/5
563/563 [==============================] - 411s 731ms/step - loss: -2.6710
Epoch 4/5
563/563 [==============================] - 411s 730ms/step - loss: -6.1618
Epoch 5/5
563/563 [==============================] - 411s 730ms/step - loss: -9.6705
<keras.src.callbacks.History at 0x7a8e840c18d0>
```

```
[ ]  loss, accuracy = model.evaluate(X_test, y_test)
     print(f"Test Accuracy: {accuracy}")
```

```
625/625 [==============================] - 43s 69ms/step - loss: -11.8148 -
Test Accuracy: 0.9056000018119812
```

MODEL COMPARISION

Models	Accuracy Percentages
Bag-of-words with MultinomialNB	0.785
TFIDF with Multinomial NB	0.83
Pretrained Naive Bayes (Text Bob)	0.789
Long Short-Term Memory (LSTM)	0.885
Recurrent neural network (RNN)	**0.90**

Limitations

- **Bag-of-words:** Does not consider word order or sequence, potentially impacting accuracy.
- **LSTM:** Performance may depend on dataset size and diversity; requires ample data for effectiveness.
- **RNN:** If a review has a lot of unnecessary details before mentioning the important point, an RNN might miss this crucial positive sentiment.

OUR NEXT STEP: BERT AND ROBERTA

We will continue to do two models in this project and evaluate the accuracy level against with previous models.

```
[ ] # Importing the necessary classes from the transformers library
    from transformers import AutoModel, AutoTokenizer

    # Initialize the model
    model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-cased", r
    metric = evaluate.load("accuracy")

    # use accuracy as evaluation metrics
    def compute_metrics(eval_pred):
        logits, labels = eval_pred
        predictions = np.argmax(logits, axis=-1)
        return metric.compute(predictions=predictions, references=labels)

    ## specify the training parameters
    training_args = TrainingArguments(output_dir="test_trainer",
                                      num_train_epochs=2,   # total 2 epochs
                                      evaluation_strategy="steps",
                                      eval_steps=100)    # evaluate every 100 steps

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_sample_train,
        eval_dataset=tokenized_sample_test,
        compute_metrics=compute_metrics,
    )

    trainer.train()
```

```
[ ] from transformers import pipeline
    from sklearn.metrics import accuracy_score

    # Load RoBERTa sentiment analysis model
    RoBERTa_sentiment = pipeline("sentiment-analysis", model="roberta-base", truncation=True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/t
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 481/481 [00:00<00:00, 12.5kB/s]
model.safetensors: 100% 499M/499M [00:07<00:00, 70.4MB/s]
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base a
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
tokenizer_config.json: 100% 25.0/25.0 [00:00<00:00, 1.69kB/s]
vocab.json: 100% 899k/899k [00:00<00:00, 1.88MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 616kB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 1.91MB/s]
```

```
[ ] # Ensure X_test is a list of strings
    X_test_text = [str(text) for text in X_test]

    # Evaluate all test instances
    # Truncation is automatically handled by the pipeline if not already set
    y_pred = [RoBERTa_sentiment(text) for text in X_test_text]

    # Assuming your function returns predictions as 'LABEL_1', 'LABEL_2', or 'LABEL_0'
    # Map predicted labels to your numeric categories
    y_pred_mapped = [1 if pred == 'LABEL_1' else (2 if pred == 'LABEL_2' else 0) for pred in y_pred]

    # y_test is accessible and correctly prepared
    accuracy = accuracy_score(y_test, y_pred_mapped)

    # Initialize performance dictionary if not already
    performance = {}
    performance["RoBERTa-sentiment"] = accuracy

    # Output the accuracy
    print("RoBERTa sentiment analysis accuracy:", accuracy)
```

THANK YOU FOR YOUR LISTENING

Questions and Answers?

Group 9 - Anvita, Rami and Onkar