Sentiment Analysis of Online Yelp Review

Mehmet Emin Ozyurt
Ilan Pinco
May 9, 2025

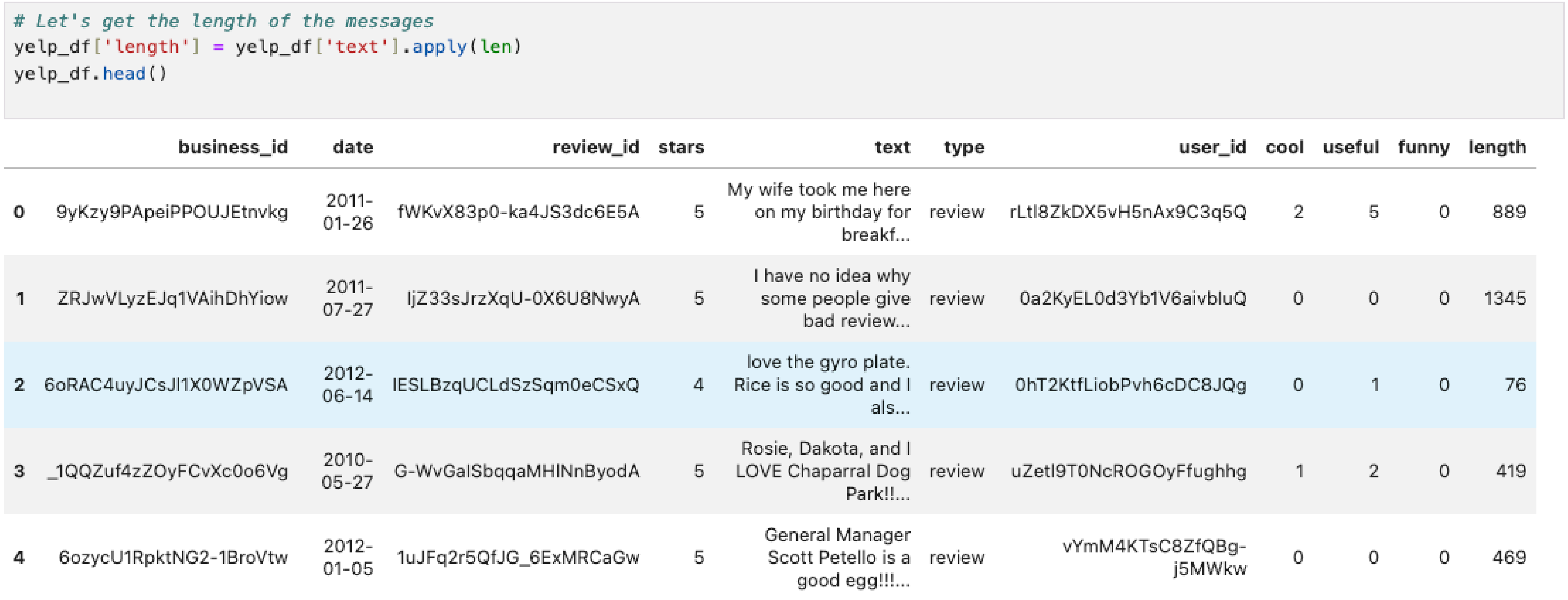
Outline

1. Introduction

- The system of five-star reviews are very common and useful.
- However, many comments about products are made outside of review boards.
- Companies often attempt to gauge public sentiment to their products through these comments but find it difficult to do so.
- The data being text-based info, and the sheer quantity are major issues.
- A Machine Learning program that can give accurate evaluations of these comments in the form of a five-star rating would allow for businesses to more efficiently determine public opinion and trends.

2. Data Overview

- The yelp dataset is from :
  - https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset
- Some samples from dataset:



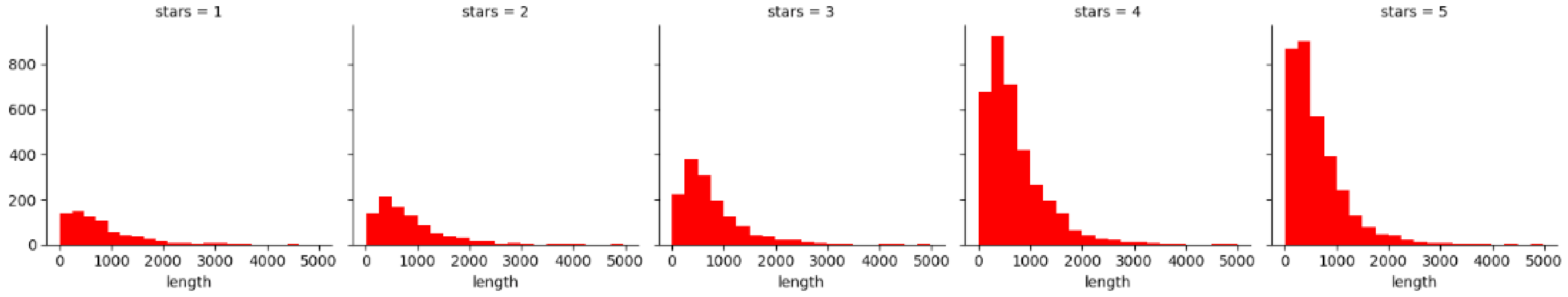
```
# Let's get the length of the messages
yelp_df['length'] = yelp_df['text'].apply(len)
yelp_df.head()
```

	business_id	date	review_id	stars	text	type	user_id	cool	useful	funny	length
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakf...	review	rLtl8ZkDX5vH5nAx9C3q5Q	2	5	0	889
1	ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some people give bad review...	review	0a2KyEL0d3Yb1V6aivbIuQ	0	0	0	1345
2	6oRAC4uyJCsJl1X0WZpVSA	2012-06-14	IESLBzqUCLdSzSqm0eCSxQ	4	love the gyro plate. Rice is so good and I als...	review	0hT2KtfLiobPvh6cDC8JQg	0	1	0	76
3	_1QQZuf4zZOyFCvXc0o6Vg	2010-05-27	G-WvGaISbqqaMHlNnByodA	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...	review	uZetl9T0NcROGOyFfughhg	1	2	0	419
4	6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	General Manager Scott Petello is a good egg!!!...	review	vYmM4KTsC8ZfQBg-j5MWkw	0	0	0	469

2. Data Overview

- The histogram plot according to their star-review

```
g = sns.FacetGrid(data=yelp_df, col='stars', col_wrap=5)
g.map(plt.hist, 'length', bins = 20, color = 'r')
plt.show()
```



- The dataset statistics

```
yelp_df.describe()
```

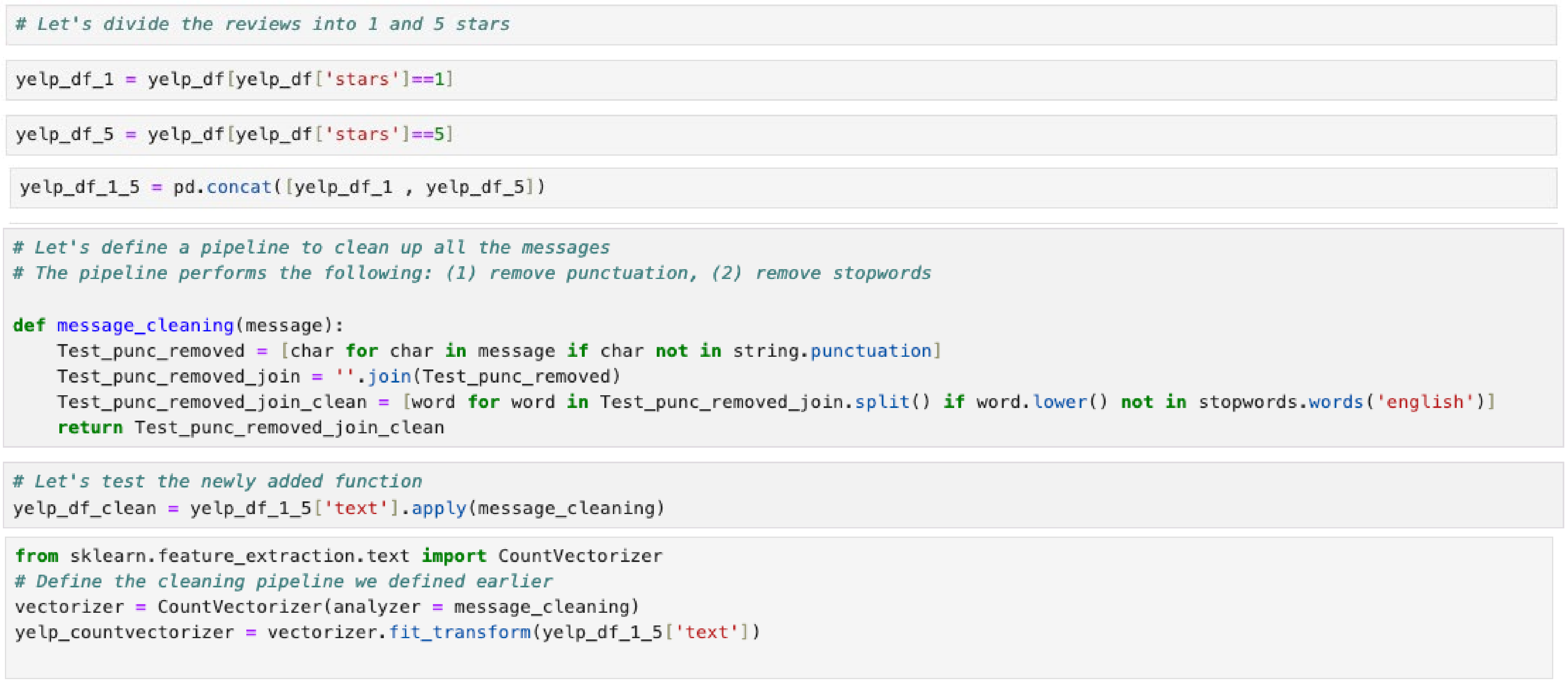
	stars	cool	useful	funny
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	3.777500	0.876800	1.409300	0.701300
std	1.214636	2.067861	2.336647	1.907942
min	1.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	0.000000
50%	4.000000	0.000000	1.000000	0.000000
75%	5.000000	1.000000	2.000000	1.000000
max	5.000000	77.000000	76.000000	57.000000

3. Data Preprocessing

- To use text data it must first be processed into a usable, and consistent form.
- First, we take the reviews/comments and remove all punctuation and stop words (and, the, etc.).
- During training we look at the most used words and create a list of a varying number of the top results. This list acts as a cypher.
- Using the cypher, an array of integers is made where each index holds the number of times the corresponding word in the cypher was used.
- This array is then multiplied with a set of weights for each of the words in the cypher. Afterwards, it is ready for evaluation by the model
- The weights are each words' TF-IDF(Term Frequency-Inverse Document Frequency), which is a weight in the form of a decimal value that tells us how important a word is within a collection of documents.
- These weights are calculated during training from the training dataset.

3. Data Preprocessing

- Text cleaning (lowercasing, removing punctuation, stopwords).
- Vectorization
  - CountVectorizer
  - TF-IDF(Term Frequency-Inverse Document Frequency)



```
# Let's divide the reviews into 1 and 5 stars
```

```
yelp_df_1 = yelp_df[yelp_df['stars']==1]
```

```
yelp_df_5 = yelp_df[yelp_df['stars']==5]
```

```
yelp_df_1_5 = pd.concat([yelp_df_1 , yelp_df_5])
```

```
# Let's define a pipeline to clean up all the messages
# The pipeline performs the following: (1) remove punctuation, (2) remove stopwords

def message_cleaning(message):
    Test_punc_removed = [char for char in message if char not in string.punctuation]
    Test_punc_removed_join = ''.join(Test_punc_removed)
    Test_punc_removed_join_clean = [word for word in Test_punc_removed_join.split() if word.lower() not in stopwords.words('english')]
    return Test_punc_removed_join_clean
```

```
# Let's test the newly added function
yelp_df_clean = yelp_df_1_5['text'].apply(message_cleaning)
```

```
from sklearn.feature_extraction.text import CountVectorizer
# Define the cleaning pipeline we defined earlier
vectorizer = CountVectorizer(analyzer = message_cleaning)
yelp_countvectorizer = vectorizer.fit_transform(yelp_df_1_5['text'])
```

3. Data Preprocessing (Example)

Original Message:

```
My wife took me here on my birthday for breakfast and it was excellent.  The weather was perfect which made sitting
outside overlooking their grounds an absolute pleasure.  Our waitress was excellent and our food arrived quickly on
the semi-busy Saturday morning.  It looked like the place fills up pretty quickly so the earlier you get here the
better.

Do yourself a favor and get their Bloody Mary.  It was phenomenal and simply the best I've ever had.  I'm pretty
sure they only use ingredients from their garden and blend them fresh when you order it.  It was amazing.

While EVERYTHING on the menu looks excellent, I had the white truffle scrambled eggs vegetable skillet and it was
tasty and delicious.  It came with 2 pieces of their griddled bread with was amazing and it absolutely made the
meal complete.  It was the best "toast" I've ever had.

Anyway, I can't wait to go back!
```

Simplified Message:

```
['wife', 'took', 'birthday', 'breakfast', 'excellent', 'weather', 'perfect', 'made', 'sitting', 'outside',
'overlooking', 'grounds', 'absolute', 'pleasure', 'waitress', 'excellent', 'food', 'arrived', 'quickly',
'semibusy', 'Saturday', 'morning', 'looked', 'like', 'place', 'fills', 'pretty', 'quickly', 'earlier', 'get',
'better', 'favor', 'get', 'Bloody', 'Mary', 'phenomenal', 'simply', 'best', 'Ive', 'ever', 'Im', 'pretty',
'sure', 'use', 'ingredients', 'garden', 'blend', 'fresh', 'order', 'amazing', 'EVERYTHING', 'menu', 'looks',
'excellent', 'white', 'truffle', 'scrambled', 'eggs', 'vegetable', 'skillet', 'tasty', 'delicious', 'came',
'2', 'pieces', 'griddled', 'bread', 'amazing', 'absolutely', 'made', 'meal', 'complete', 'best', 'toast',
'Ive', 'ever', 'Anyway', 'cant', 'wait', 'go', 'back']
```

4. Model Building

- Split data into train/test sets.
  - 80% of the dataset is used for training
  - 20% is used for testing
- Model Used:

1. Naive Bayes

- Well-suited for text classification due to its simplicity and efficiency.
- Performs well on high-dimensional data like TF-IDF vectors, despite strong independence assumptions.

2. Logistic Regression

- Offers good baseline performance for multi-class classification tasks.
- Interpretable and handles large feature spaces (e.g., from text vectorization) effectively.

3. Support Vector Machine (SVM)

- Effective in high-dimensional spaces and works well with clear margin separation.
- Particularly good for text classification problems with sparse data.

4. Random Forest

- Handles non-linear relationships and interactions between features.
- Provides feature importance metrics and performs well with unstructured text after vectorization.

5. Results & Evaluation

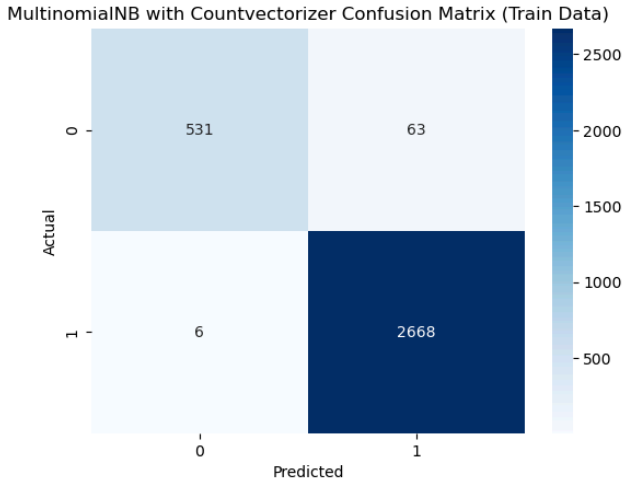
Naive Bayes with CountVectorizer on Train/Test Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

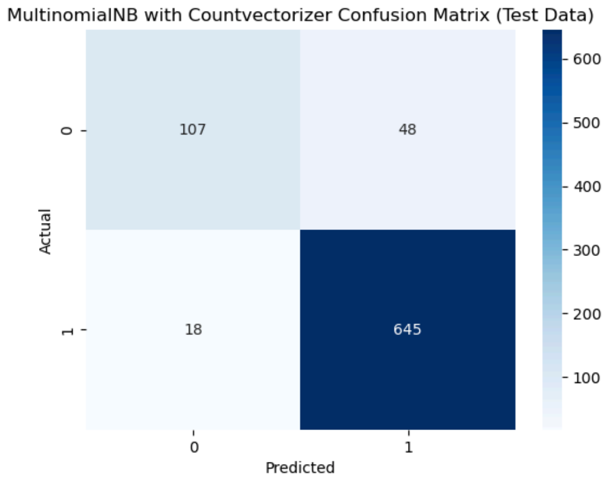
```
from sklearn.naive_bayes import MultinomialNB

NB_classifier = MultinomialNB()
NB_classifier.fit(X_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
y_predict_train = NB_classifier.predict(X_train)
y_predict_train
cm = confusion_matrix(y_train, y_predict_train)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_title('MultinomialNB with Countvectorizer Confusion Matrix (Train Data)')
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```

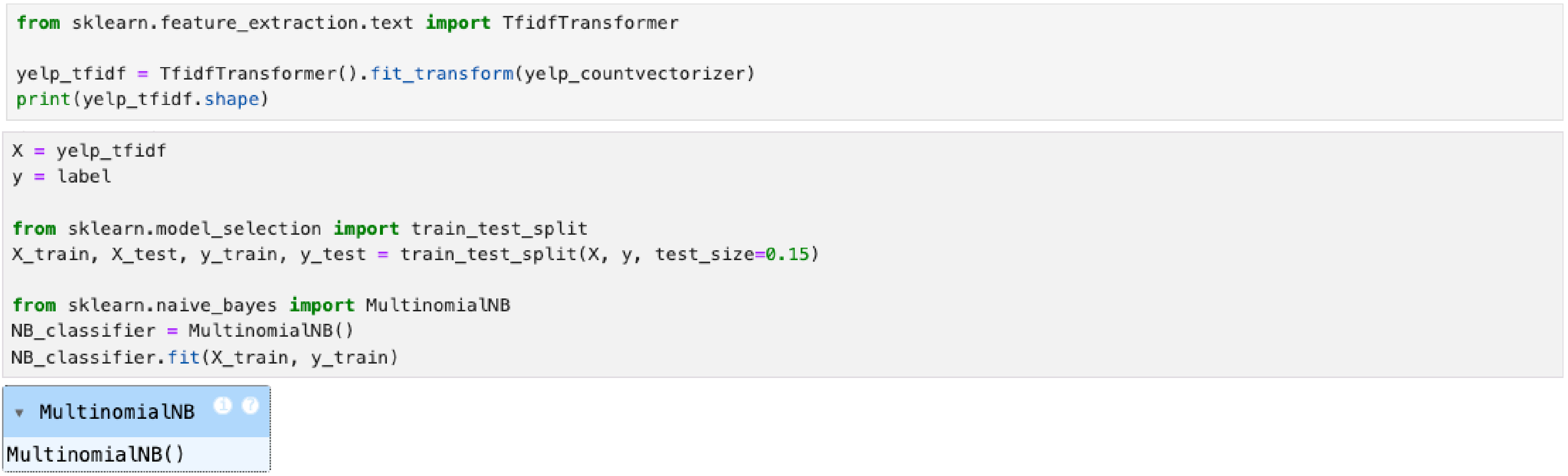


```
# Predicting the Test set results
y_predict_test = NB_classifier.predict(X_test)
cm = confusion_matrix(y_test, y_predict_test)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_title('MultinomialNB with Countvectorizer Confusion Matrix (Test Data)')
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
```



5. Results & Evaluation

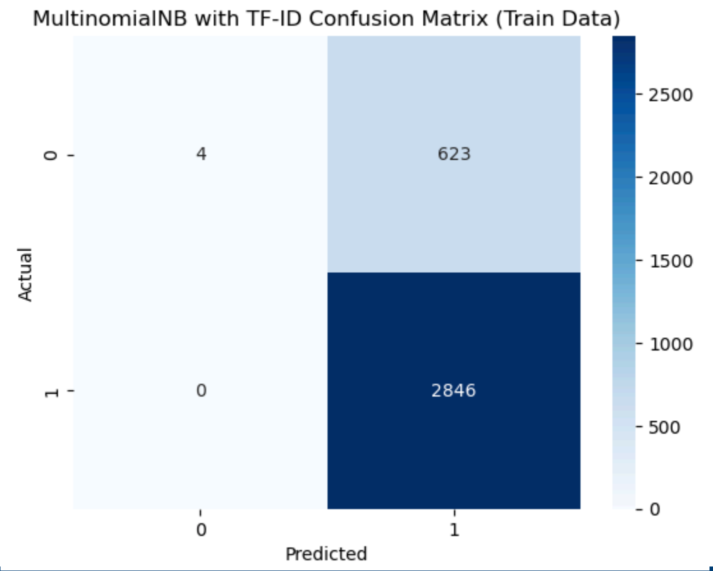
Naive Bayes with TF-IDF on Train/Test Data

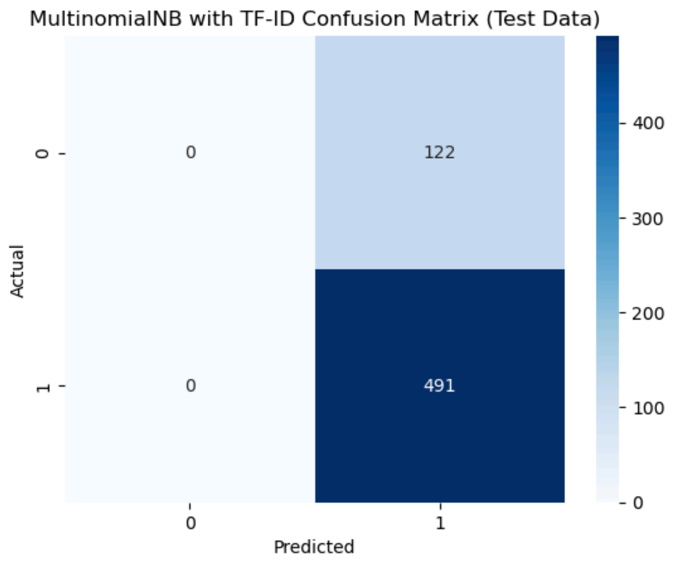


```
from sklearn.feature_extraction.text import TfidfTransformer

yelp_tfidf = TfidfTransformer().fit_transform(yelp_countvectorizer)
print(yelp_tfidf.shape)
```

```
X = yelp_tfidf
y = label

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)

from sklearn.naive_bayes import MultinomialNB
NB_classifier = MultinomialNB()
NB_classifier.fit(X_train, y_train)
```

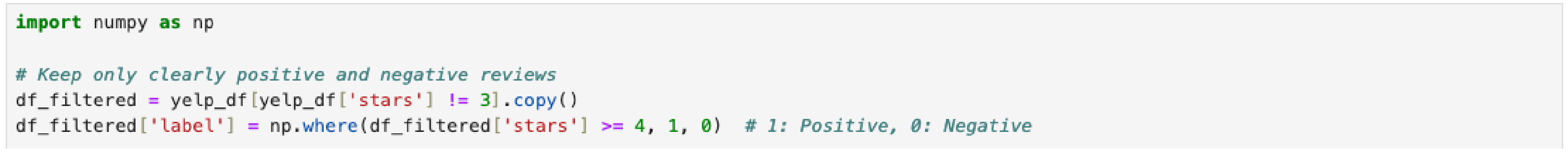
```
▾ MultinomialNB
MultinomialNB()
```



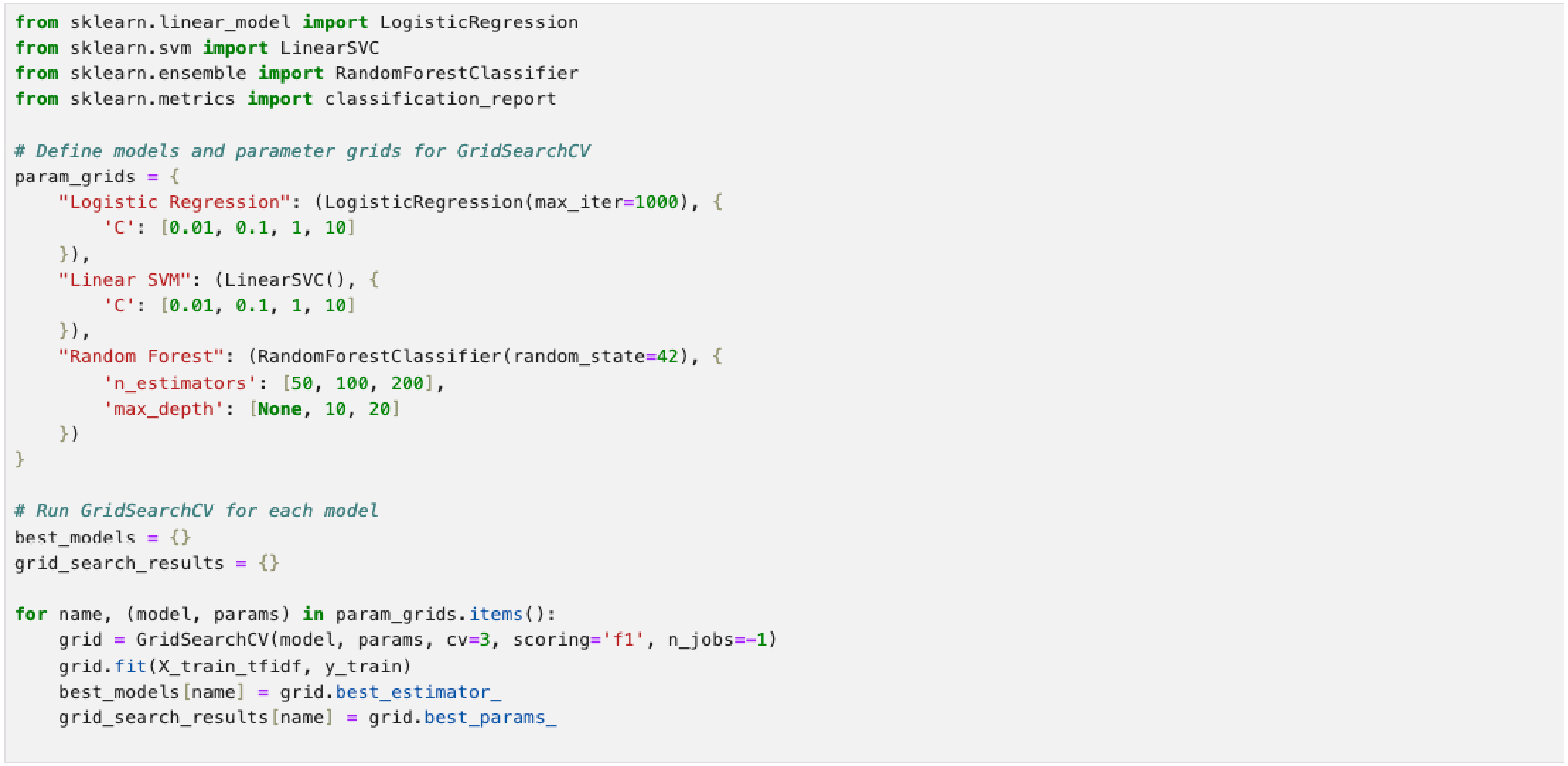


5. Results & Evaluation

Logistic Regression, Linear SVM, & Random Forest Models



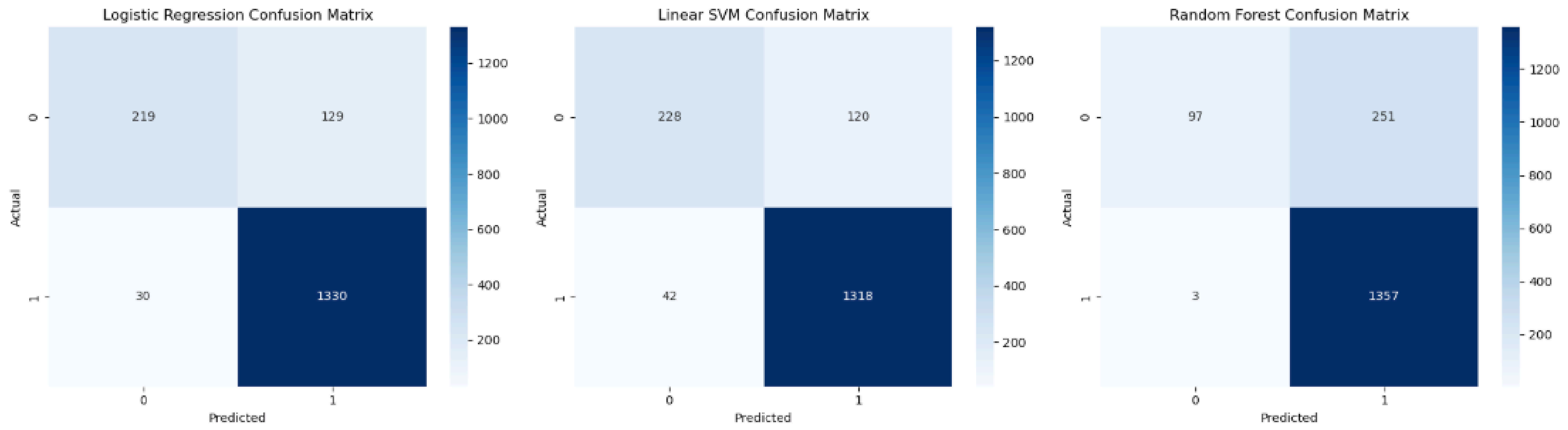
```python
import numpy as np

# Keep only clearly positive and negative reviews
df_filtered = yelp_df[yelp_df['stars'] != 3].copy()
df_filtered['label'] = np.where(df_filtered['stars'] >= 4, 1, 0)  # 1: Positive, 0: Negative
```

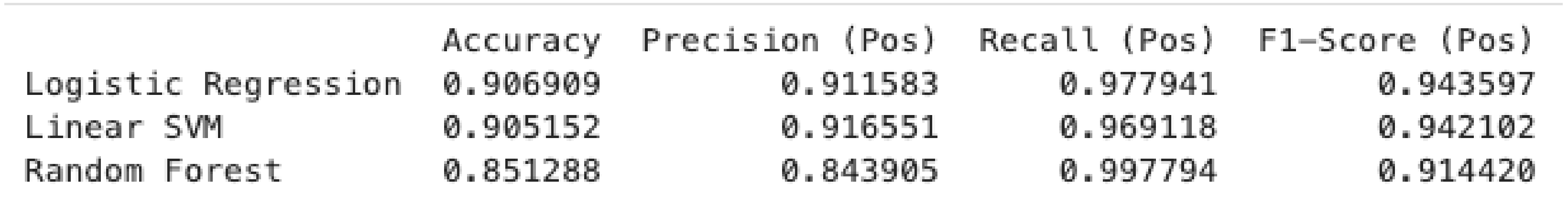


```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Define models and parameter grids for GridSearchCV
param_grids = {
    "Logistic Regression": (LogisticRegression(max_iter=1000), {
        'C': [0.01, 0.1, 1, 10]
    }),
    "Linear SVM": (LinearSVC(), {
        'C': [0.01, 0.1, 1, 10]
    }),
    "Random Forest": (RandomForestClassifier(random_state=42), {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20]
    })
}

# Run GridSearchCV for each model
best_models = {}
grid_search_results = {}

for name, (model, params) in param_grids.items():
    grid = GridSearchCV(model, params, cv=3, scoring='f1', n_jobs=-1)
    grid.fit(X_train_tfidf, y_train)
    best_models[name] = grid.best_estimator_
    grid_search_results[name] = grid.best_params_
```

5. Results & Evaluation

Logistic Regression, Linear SVM, & Random Forest Models





```
                     Accuracy  Precision (Pos)  Recall (Pos)  F1-Score (Pos)
Logistic Regression  0.906909         0.911583      0.977941        0.943597
Linear SVM           0.905152         0.916551      0.969118        0.942102
Random Forest        0.851288         0.843905      0.997794        0.914420
```

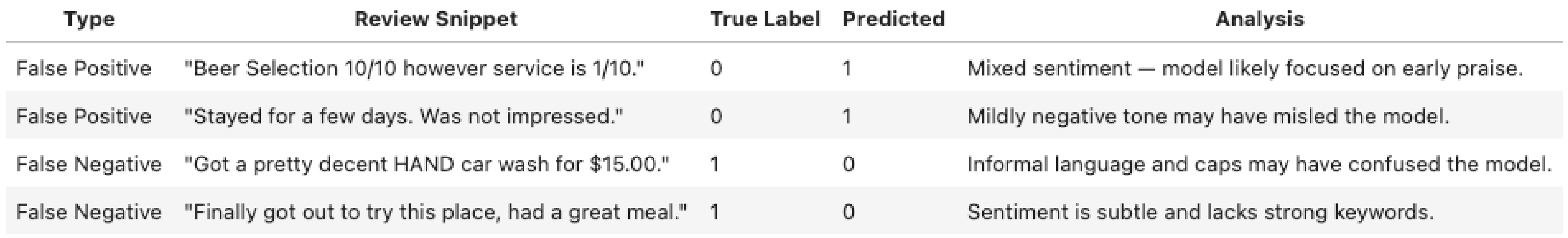
Best Model: Logistic Regression

- Highest accuracy and F1-score.
- Efficient for high-dimensional text data like TF-IDF vectors.

5. Results & Evaluation

Misclassified Review Analysis

- After running our data through the models, we analyzed the misclassifications of the logistic regression model.
- The model seemed to have "weak spots" when it came to mixed sentiment reviews, subtle or polite complaints, and informal language.

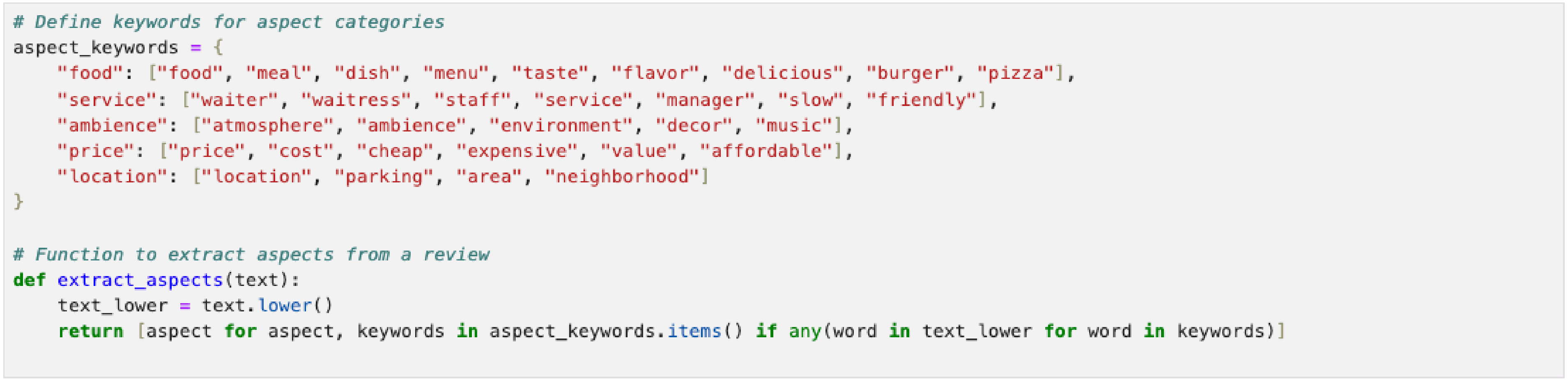


Type	Review Snippet	True Label	Predicted	Analysis
False Positive	"Beer Selection 10/10 however service is 1/10."	0	1	Mixed sentiment — model likely focused on early praise.
False Positive	"Stayed for a few days. Was not impressed."	0	1	Mildly negative tone may have misled the model.
False Negative	"Got a pretty decent HAND car wash for $15.00."	1	0	Informal language and caps may have confused the model.
False Negative	"Finally got out to try this place, had a great meal."	1	0	Sentiment is subtle and lacks strong keywords.

5. Results & Evaluation

Adding **aspect-based sentiment analysis** can be a powerful and unique extension to models. Instead of classifying the *overall* sentiment of a review, this approach will break the sentiment down by aspects such as:

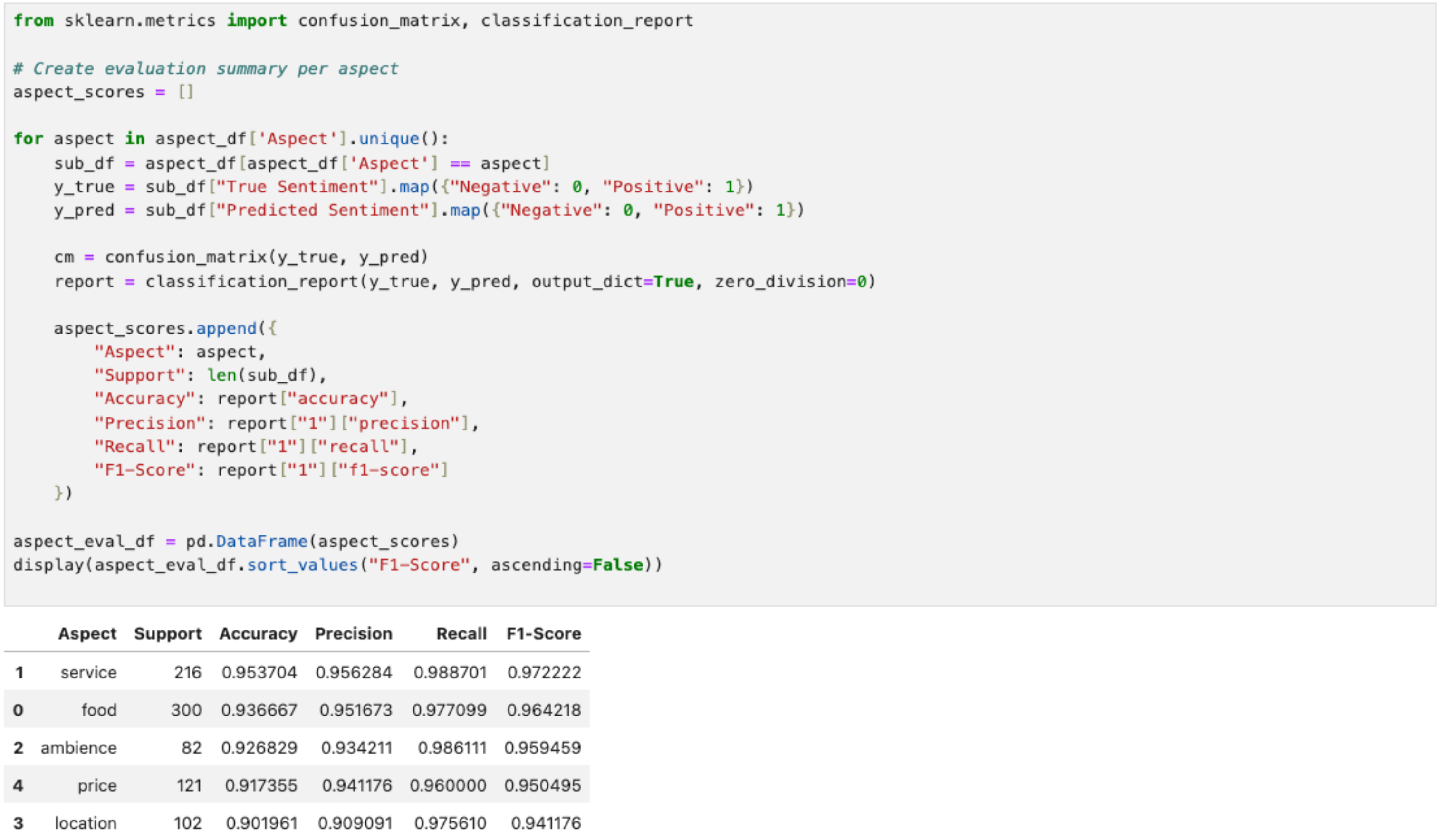
- **Food**
- **Service**
- **Ambience**
- **Price**
- **Location**



```python
# Define keywords for aspect categories
aspect_keywords = {
    "food": ["food", "meal", "dish", "menu", "taste", "flavor", "delicious", "burger", "pizza"],
    "service": ["waiter", "waitress", "staff", "service", "manager", "slow", "friendly"],
    "ambience": ["atmosphere", "ambience", "environment", "decor", "music"],
    "price": ["price", "cost", "cheap", "expensive", "value", "affordable"],
    "location": ["location", "parking", "area", "neighborhood"]
}

# Function to extract aspects from a review
def extract_aspects(text):
    text_lower = text.lower()
    return [aspect for aspect, keywords in aspect_keywords.items() if any(word in text_lower for word in keywords)]
```

5. Results & Evaluation

Scores of aspect-based sentiment analysis

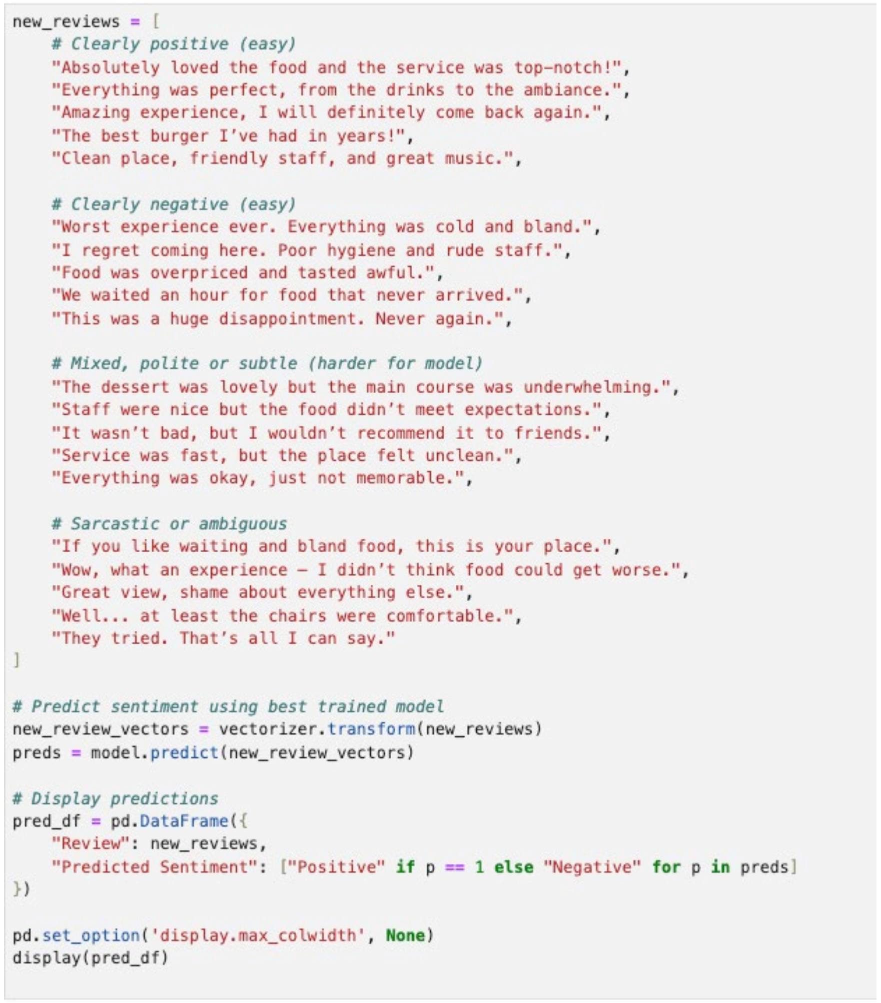


```python
from sklearn.metrics import confusion_matrix, classification_report

# Create evaluation summary per aspect
aspect_scores = []

for aspect in aspect_df['Aspect'].unique():
    sub_df = aspect_df[aspect_df['Aspect'] == aspect]
    y_true = sub_df["True Sentiment"].map({"Negative": 0, "Positive": 1})
    y_pred = sub_df["Predicted Sentiment"].map({"Negative": 0, "Positive": 1})

    cm = confusion_matrix(y_true, y_pred)
    report = classification_report(y_true, y_pred, output_dict=True, zero_division=0)

    aspect_scores.append({
        "Aspect": aspect,
        "Support": len(sub_df),
        "Accuracy": report["accuracy"],
        "Precision": report["1"]["precision"],
        "Recall": report["1"]["recall"],
        "F1-Score": report["1"]["f1-score"]
    })

aspect_eval_df = pd.DataFrame(aspect_scores)
display(aspect_eval_df.sort_values("F1-Score", ascending=False))
```

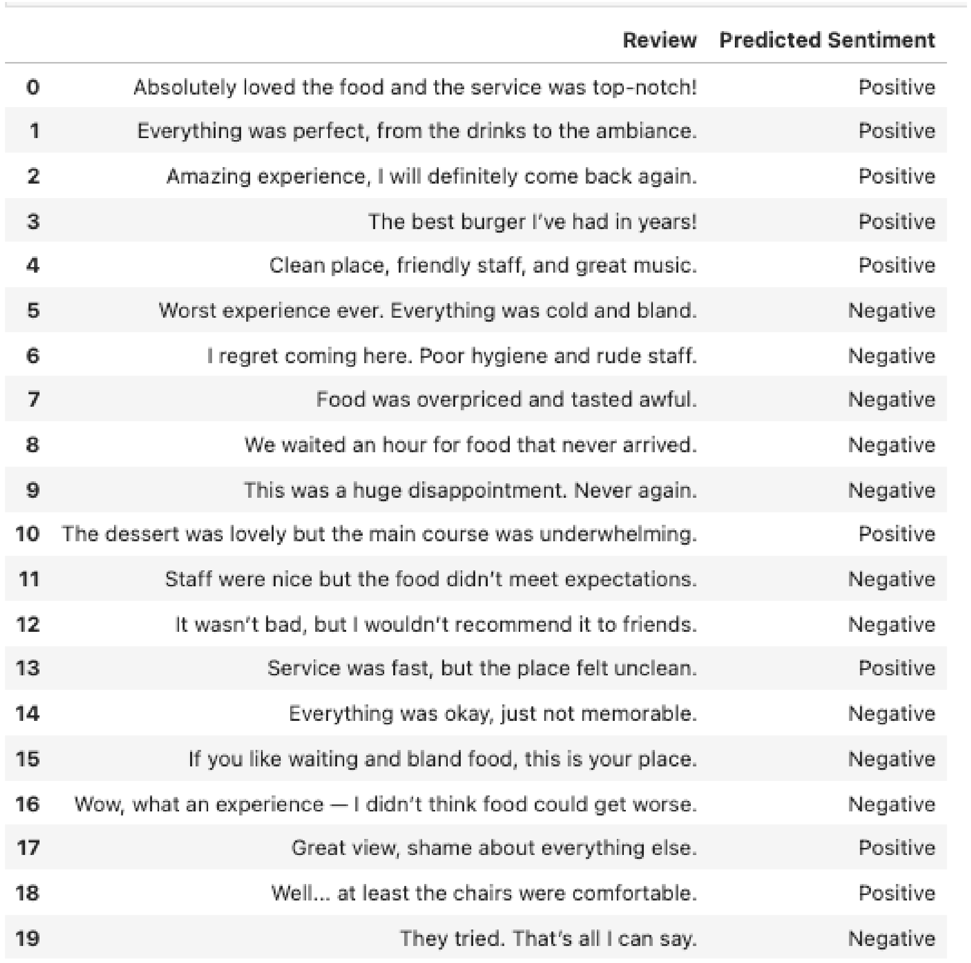
	Aspect	Support	Accuracy	Precision	Recall	F1-Score
1	service	216	0.953704	0.956284	0.988701	0.972222
0	food	300	0.936667	0.951673	0.977099	0.964218
2	ambience	82	0.926829	0.934211	0.986111	0.959459
4	price	121	0.917355	0.941176	0.960000	0.950495
3	location	102	0.901961	0.909091	0.975610	0.941176

5. Results & Evaluation

Model on "unlabeled" reviews.



```
new_reviews = [
    # Clearly positive (easy)
    "Absolutely loved the food and the service was top-notch!",
    "Everything was perfect, from the drinks to the ambiance.",
    "Amazing experience, I will definitely come back again.",
    "The best burger I've had in years!",
    "Clean place, friendly staff, and great music.",

    # Clearly negative (easy)
    "Worst experience ever. Everything was cold and bland.",
    "I regret coming here. Poor hygiene and rude staff.",
    "Food was overpriced and tasted awful.",
    "We waited an hour for food that never arrived.",
    "This was a huge disappointment. Never again.",

    # Mixed, polite or subtle (harder for model)
    "The dessert was lovely but the main course was underwhelming.",
    "Staff were nice but the food didn't meet expectations.",
    "It wasn't bad, but I wouldn't recommend it to friends.",
    "Service was fast, but the place felt unclean.",
    "Everything was okay, just not memorable.",

    # Sarcastic or ambiguous
    "If you like waiting and bland food, this is your place.",
    "Wow, what an experience — I didn't think food could get worse.",
    "Great view, shame about everything else.",
    "Well... at least the chairs were comfortable.",
    "They tried. That's all I can say."
]

# Predict sentiment using best trained model
new_review_vectors = vectorizer.transform(new_reviews)
preds = model.predict(new_review_vectors)

# Display predictions
pred_df = pd.DataFrame({
    "Review": new_reviews,
    "Predicted Sentiment": ["Positive" if p == 1 else "Negative" for p in preds]
})

pd.set_option('display.max_colwidth', None)
display(pred_df)
```



	Review	Predicted Sentiment
0	Absolutely loved the food and the service was top-notch!	Positive
1	Everything was perfect, from the drinks to the ambiance.	Positive
2	Amazing experience, I will definitely come back again.	Positive
3	The best burger I've had in years!	Positive
4	Clean place, friendly staff, and great music.	Positive
5	Worst experience ever. Everything was cold and bland.	Negative
6	I regret coming here. Poor hygiene and rude staff.	Negative
7	Food was overpriced and tasted awful.	Negative
8	We waited an hour for food that never arrived.	Negative
9	This was a huge disappointment. Never again.	Negative
10	The dessert was lovely but the main course was underwhelming.	Positive
11	Staff were nice but the food didn't meet expectations.	Negative
12	It wasn't bad, but I wouldn't recommend it to friends.	Negative
13	Service was fast, but the place felt unclean.	Positive
14	Everything was okay, just not memorable.	Negative
15	If you like waiting and bland food, this is your place.	Negative
16	Wow, what an experience — I didn't think food could get worse.	Negative
17	Great view, shame about everything else.	Positive
18	Well... at least the chairs were comfortable.	Positive
19	They tried. That's all I can say.	Negative

6. Conclusion & Future Work

Conclusion:

- Our study demonstrates that machine learning models, particularly Logistic Regression with TF-IDF, are effective for multi-class sentiment classification of Yelp reviews.
- Common challenges include handling mixed sentiments, sarcasm, and domain-specific expressions.

Future Work:

- Enhance aspect-based sentiment analysis to capture granular sentiments for categories like food, service, and ambience.
- Expand the dataset by scraping real-time reviews to improve generalization.
- Integrate neutral sentiment as a distinct class to improve classification robustness.
- Explore deep learning methods (e.g., BERT, LSTM) for richer contextual understanding.

THANK YOU ☺

Questions?