In [1]:

```
import json
import pandas as pd

data_file_path = 'C:/Users/ankit/yelp_merged_data.csv'
checkin_df = pd.read_csv(data_file_path)

# Geting number of columns
num_columns = checkin_df.shape[1]

print(f"The number of columns in the dataset is: {num_columns}")
# Getting all the columns in the DataFrame
print("Name of the Columns in the dataset:")
for column in checkin_df.columns:
    print(column)

# Printing shape and details of the dataset
print("Shape of the dataset:")
print(checkin_df.shape)

# Printing Column Names:
print("Column names:")
print(checkin_df.columns)

# Printing Column Name with Data Type
print("Datatype of each column:")
print(checkin_df.dtypes)

# Printing first 5 rows of the dataset
print("Few dataset entries:")
print(checkin_df.head())

# Printing summart of the dataset
checkin_df.describe(include='all')
```

```
1  {"Monday": "6:30-20:30", "Tuesday": "6:30-20:3...
2                                                NaN
3  {"Tuesday": "11:0-21:0", "Wednesday": "11:0-21...
4  {"Monday": "0:0-0:0", "Friday": "11:0-17:0", "...

[5 rows x 21 columns]
```

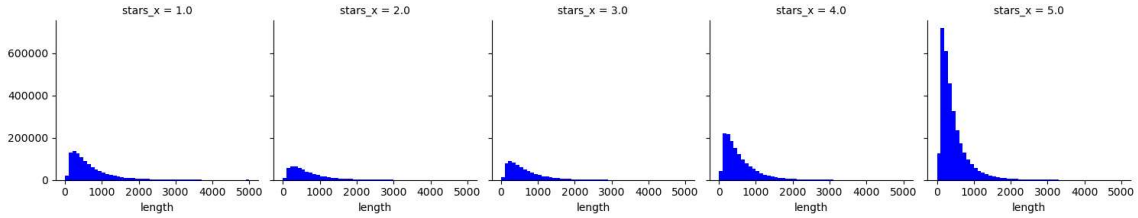
Out[1]:

	review_id	user_id	business_id	st
count	6990280	6990280	6990280	6.99028(
unique	6990280	1987929	150346	
top	KU_O5udG6zpxOg-VcAEodg	_BcWyKQL16ndpBdggh2kNA	_ab50qdWOk0DdB6XOrBitw	
freq	1	3048	7673	

In [3]:

```
import seaborn as sns
import matplotlib.pyplot as plt
#Adding and creating a new column named "Length" to count the number of
checkin_df['length'] = checkin_df['text'].apply(len)
checkin_df.head()

graph = sns.FacetGrid(data=checkin_df,col='stars_x')
graph.map(plt.hist,'length',bins=50,color='blue')
```

Out[3]: <seaborn.axisgrid.FacetGrid at 0x2154b05f410>



In [10]:

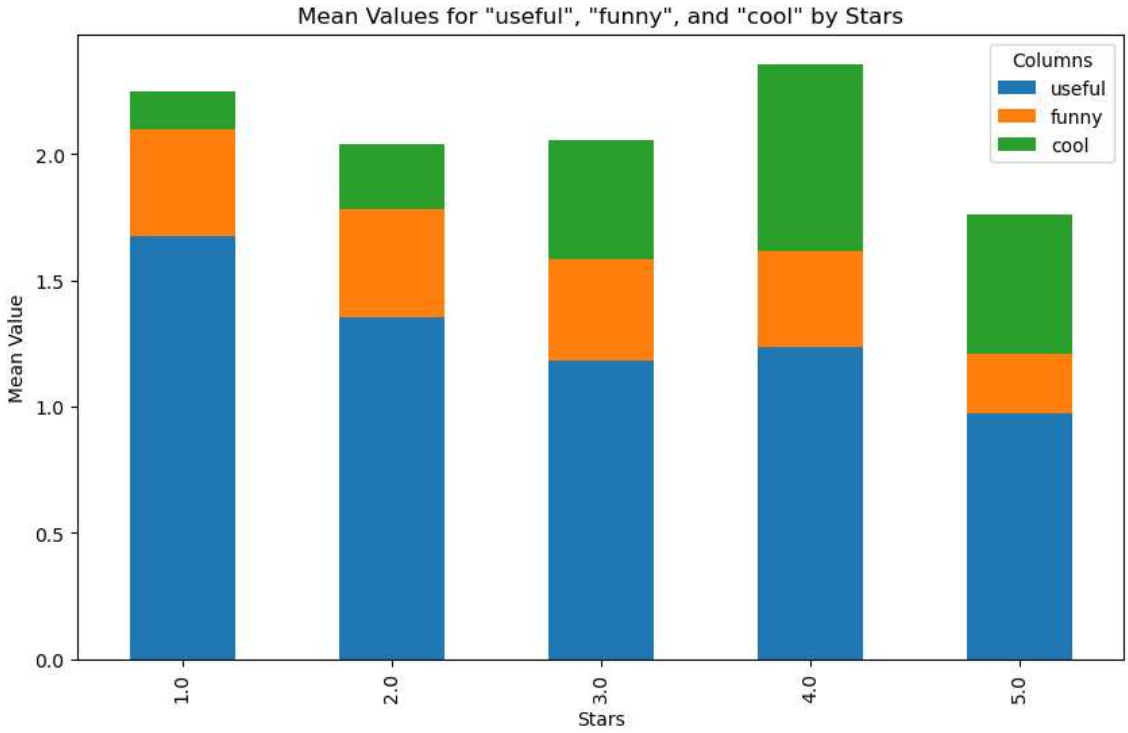
```
st_mean = checkin_df.groupby('stars_x').mean(numeric_only=True)
st_mean
```

Out[10]:

stars_x	useful	funny	cool	postal code	latitude	longitude	stars_y	review_count
1.0	1.673746	0.422610	0.154404	NaN	36.071437	-89.744251	3.009984	201.685561
2.0	1.354246	0.426679	0.258720	NaN	36.221857	-89.271493	3.434126	338.673870
3.0	1.181042	0.400245	0.472440	NaN	36.227045	-88.885623	3.610429	418.176550
4.0	1.234359	0.380260	0.742942	NaN	36.184931	-88.972196	3.799850	443.546784
5.0	0.972549	0.237988	0.548700	NaN	35.695603	-90.320310	4.058462	385.808362

In [11]:

```
import matplotlib.pyplot as plt

selected_columns = ['useful', 'funny', 'cool']
stval_selected = st_mean[selected_columns]

stval_selected.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Mean Values for "useful", "funny", and "cool" by Stars')
plt.xlabel('Stars')
plt.ylabel('Mean Value')
plt.legend(title='Columns', bbox_to_anchor=(1, 1))

plt.show()
```



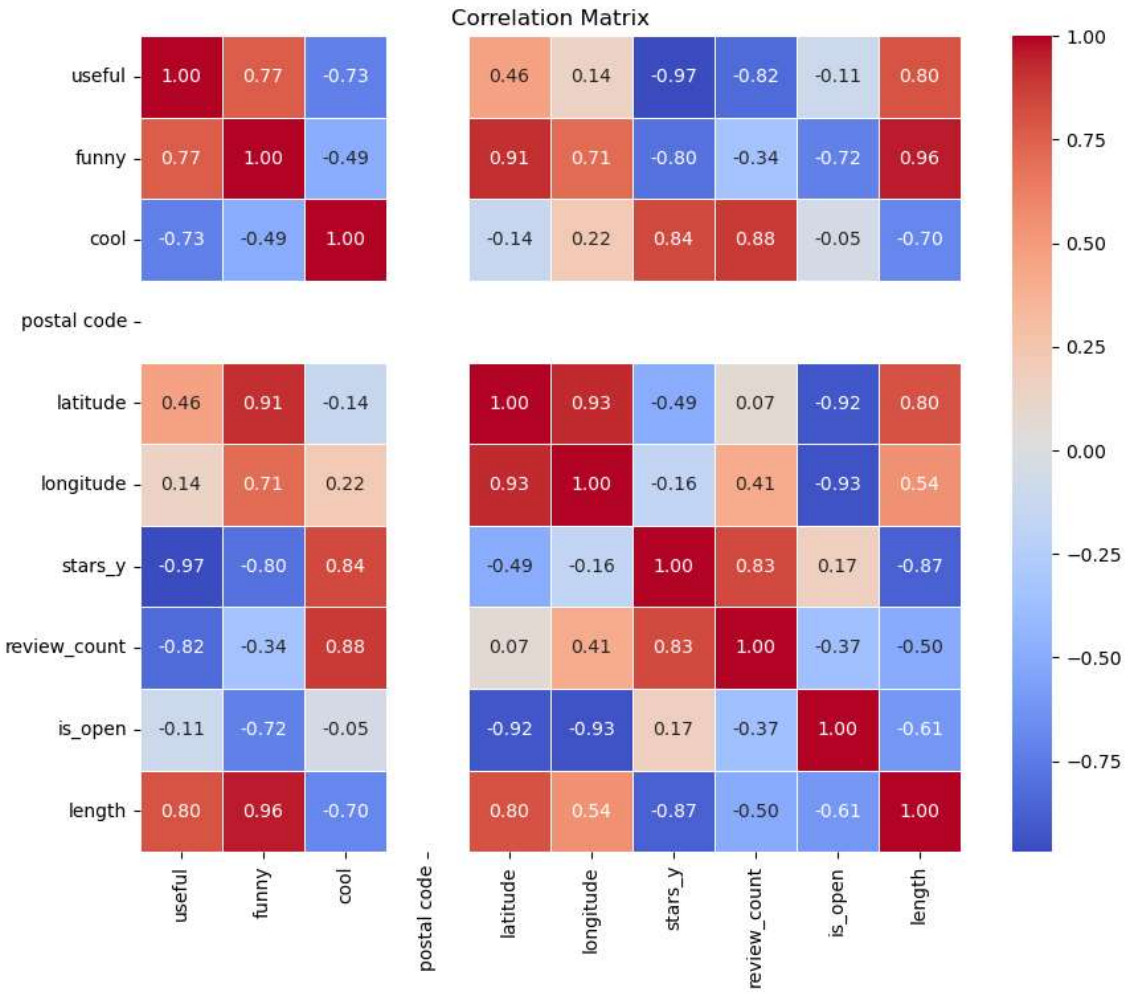
In [12]:
```
1 st_mean.corr()
```

Out[12]:

	useful	funny	cool	postal code	latitude	longitude	stars_y	review
useful	1.000000	0.768755	-0.729256	NaN	0.458122	0.141436	-0.968761	-0.8
funny	0.768755	1.000000	-0.485747	NaN	0.913415	0.705389	-0.796918	-0.3
cool	-0.729256	-0.485747	1.000000	NaN	-0.135952	0.215737	0.835427	0.8
postal code	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
latitude	0.458122	0.913415	-0.135952	NaN	1.000000	0.927746	-0.486565	0.0
longitude	0.141436	0.705389	0.215737	NaN	0.927746	1.000000	-0.162000	0.4
stars_y	-0.968761	-0.796918	0.835427	NaN	-0.486565	-0.162000	1.000000	0.8
review_count	-0.824346	-0.344793	0.883743	NaN	0.066983	0.408990	0.833577	1.0
is_open	-0.106978	-0.715643	-0.049289	NaN	-0.920341	-0.934607	0.171321	-0.3
length	0.799799	0.960635	-0.695158	NaN	0.804144	0.539141	-0.874715	-0.5

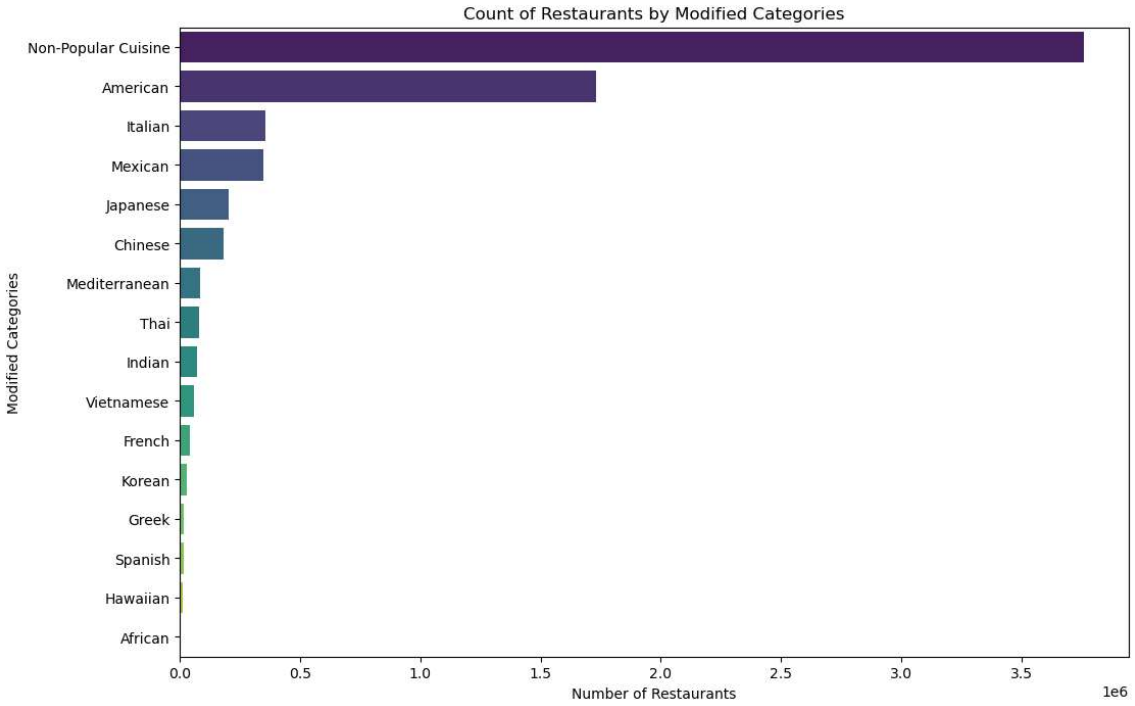
In [13]:

```
import seaborn as sns
import matplotlib.pyplot as plt

correlation_matrix = st_mean.corr()

# Creating a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
plt.title('Correlation Matrix')
plt.show()
```



In [50]:

```
# Create the modified 'categories' column based on the mapping
checkin_df['modified_categories'] = checkin_df['categories'].apply(
    lambda x: 'American' if 'restaurant' in x.lower() and 'american' in
              'Mexican' if 'restaurant' in x.lower() and 'mexican' in x.
              'Italian' if 'restaurant' in x.lower() and 'italian' in x.
              'Japanese' if 'restaurant' in x.lower() and 'japanese' in
              'Chinese' if 'restaurant' in x.lower() and 'chinese' in x.
              'Thai' if 'restaurant' in x.lower() and 'thai' in x.lower(
              'Mediterranean' if 'restaurant' in x.lower() and 'mediterr
              'French' if 'restaurant' in x.lower() and 'french' in x.lo
              'Vietnamese' if 'restaurant' in x.lower() and 'vietnamese'
              'Greek' if 'restaurant' in x.lower() and 'greek' in x.lowe
              'Indian' if 'restaurant' in x.lower() and 'indian' in x.lo
              'Korean' if 'restaurant' in x.lower() and 'korean' in x.lo
              'Hawaiian' if 'restaurant' in x.lower() and 'hawaiian' in
              'African' if 'restaurant' in x.lower() and 'african' in x.
              'Spanish' if 'restaurant' in x.lower() and 'spanish' in x.
              'Middle_eastern' if 'restaurant' in x.lower() and 'middle_
              'Non-Popular Cuisine'
)


modified_categories_counts = checkin_df['modified_categories'].value_co

modified_categories_counts = modified_categories_counts.sort_values(asc

# Plot the graph of value counts
plt.figure(figsize=(12, 8))
sns.barplot(x=modified_categories_counts, y=modified_categories_counts.
plt.title('Count of Restaurants by Modified Categories')
plt.xlabel('Number of Restaurants')
plt.ylabel('Modified Categories')
plt.show()
# Count the categories in the 'modified_categories' column
#modified_categories_counts = checkin_df['modified_categories'].value_c

#for category, count in modified_categories_counts.items():
    #print(f"{category}: {count} restaurants")

```



In [81]:

```
# CLASSIFICATION
checkin_df['review_category'] = pd.cut(checkin_df['stars_x'], bins=[0,

# Filter the DataFrame based on the new 'review_category' column
data_classes = checkin_df[checkin_df['review_category'].isin(['bad', 'a
data_classes.head()
print(data_classes.shape)

# Separate the dataset into X and Y for prediction
#x = data_classes['text'].iloc[:10000]
x = data_classes['text']
y = data_classes['review_category']
#y = data_classes['review_category'].iloc[:10000]

print(x.head())
print(y.head())
```

```
(6990280, 24)
0    If you decide to eat here, just be aware it is...
1    I've taken a lot of spin classes over the year...
2    Family diner. Had the buffet. Eclectic assortm...
3    Wow!  Yummy, different,  delicious.   Our favo...
4    Cute interior and owner (?) gave us tour of up...
Name: text, dtype: object
0    average
1       good
2    average
3       good
4       good
Name: review_category, dtype: category
Categories (3, object): ['bad' < 'average' < 'good']
```

In [70]:
```
import nltk
from nltk.corpus import stopwords
import string
from nltk.tokenize import sent_tokenize
sent_tokenizer = sent_tokenize
from sklearn.feature_extraction.text import CountVectorizer

# CLEANING THE REVIEWS - REMOVAL OF STOPWORDS AND PUNCTUATION
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stop
```

In [77]:
```
import nltk
from nltk.corpus import stopwords
import string
from nltk.tokenize import sent_tokenize
from sklearn.feature_extraction.text import CountVectorizer


x = checkin_df['text'].head(10000)
#x = checkin_df['text'] run this line of code for the whole data set, w
# but it is taking forever to complete preprocessing in our huge datase

# CLEANING THE REVIEWS - REMOVAL OF STOPWORDS AND PUNCTUATION
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stop

# CONVERTING THE WORDS INTO A VECTOR
vocab = CountVectorizer(analyzer=text_process).fit(x)
print(len(vocab.vocabulary_))

r0 = x.iloc[0]
vocab0 = vocab.transform([r0])
```

```
35071
```

In [78]:
```
# Transforming the features using the vocabulary
transformed_x = vocab.transform(x)

# Displaying information about the sparse matrix:
print("The shape of the sparse matrix is: ", transformed_x.shape)
print("Number of non-zero occurrences: ", transformed_x.nnz)

# density of the matrix:
density_percentage = (transformed_x.nnz / (transformed_x.shape[0] * tra
print("Density of the matrix: {:.2f}%".format(density_percentage))
```

```
The shape of the sparse matrix is:  (10000, 35071)
Number of non-zero occurrences:  457423
Density of the matrix: 0.13%
```

In [92]:
```
from sklearn.model_selection import train_test_split
# SPLITTING THE DATASET INTO TRAINING SET AND TESTING SET
x_subset = x[:10000]
y_subset = y[:10000]
x_train, x_test, y_train, y_test = train_test_split(x_subset, y_subset,
```

In [90]:
```
vectorizer = CountVectorizer()

# Fit and transform the training data
x_train_vectorized = vectorizer.fit_transform(x_train)

# Transform the test data
x_test_vectorized = vectorizer.transform(x_test)
```

In [93]:

```
# Multinomial Naive Bayes
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 confusion_matrix, accuracy_score, classific



tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data
x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)

# Transform the test data
x_test_tfidf = tfidf_vectorizer.transform(x_test)

# Creating and training the Multinomial Naive Bayes classifier
mnb_classifier = MultinomialNB()
mnb_classifier.fit(x_train_tfidf, y_train)

# Making predictions on the test set
predictions_mnb = mnb_classifier.predict(x_test_tfidf)

# Evaluating the Multinomial Naive Bayes classifier
print("Confusion Matrix for Multinomial Naive Bayes:")
print(confusion_matrix(y_test, predictions_mnb))

accuracy = accuracy_score(y_test, predictions_mnb) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_mnb))

```

```
Confusion Matrix for Multinomial Naive Bayes:
[[   0    0  213]
 [   0    1  364]
 [   0    0 1422]]
Accuracy Score: 71.15%
Classification Report:
              precision    recall  f1-score   support

     average       0.00      0.00      0.00       213
         bad       1.00      0.00      0.01       365
        good       0.71      1.00      0.83      1422

    accuracy                           0.71      2000
   macro avg       0.57      0.33      0.28      2000
weighted avg       0.69      0.71      0.59      2000
```

```
C:\Users\ankit\anaconda3\Lib\site-packages\sklearn\metrics\_classificatio
n.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ankit\anaconda3\Lib\site-packages\sklearn\metrics\_classificatio
n.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ankit\anaconda3\Lib\site-packages\sklearn\metrics\_classificatio
n.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [95]:

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=
rf_classifier.fit(x_train_tfidf, y_train)


predictions_rf = rf_classifier.predict(x_test_tfidf)

# Evaluating the Random Forest Classifier
print("Confusion Matrix for Random Forest Classifier:")
print(confusion_matrix(y_test, predictions_rf))

accuracy = accuracy_score(y_test, predictions_rf) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_rf))
```

```
Confusion Matrix for Random Forest Classifier:
[[   1   10  202]
 [   0  113  252]
 [   0    1 1421]]
Accuracy Score: 76.75%
Classification Report:
              precision    recall  f1-score   support

     average       1.00      0.00      0.01       213
         bad       0.91      0.31      0.46       365
        good       0.76      1.00      0.86      1422

    accuracy                           0.77      2000
   macro avg       0.89      0.44      0.44      2000
weighted avg       0.81      0.77      0.70      2000
```

In [96]:

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=101)
dt_classifier.fit(x_train_tfidf, y_train)


predictions_dt = dt_classifier.predict(x_test_tfidf)


print("Confusion Matrix for Decision Tree Classifier:")
print(confusion_matrix(y_test, predictions_dt))

accuracy = accuracy_score(y_test, predictions_dt) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_dt))
```

```
Confusion Matrix for Decision Tree Classifier:
[[  53   38  122]
 [  56  184  125]
 [ 136  148 1138]]
Accuracy Score: 68.75%
Classification Report:
              precision    recall  f1-score   support

     average       0.22      0.25      0.23       213
         bad       0.50      0.50      0.50       365
        good       0.82      0.80      0.81      1422

    accuracy                           0.69      2000
   macro avg       0.51      0.52      0.51      2000
weighted avg       0.70      0.69      0.69      2000
```

In [97]:

```
# Support Vector Machine
from sklearn.svm import SVC
svm_classifier = SVC(kernel='linear', random_state=101)
svm_classifier.fit(x_train_tfidf, y_train)


predictions_svm = svm_classifier.predict(x_test_tfidf)

# Evaluating the Support Vector Machine (SVM) Classifier
print("Confusion Matrix for Support Vector Machine (SVM) Classifier:")
print(confusion_matrix(y_test, predictions_svm))

accuracy = accuracy_score(y_test, predictions_svm) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_svm))
```

```
Confusion Matrix for Support Vector Machine (SVM) Classifier:
[[  41   43  129]
 [  18  289   58]
 [  27   16 1379]]
Accuracy Score: 85.45%
Classification Report:
              precision    recall  f1-score   support

     average       0.48      0.19      0.27       213
         bad       0.83      0.79      0.81       365
        good       0.88      0.97      0.92      1422

    accuracy                           0.85      2000
   macro avg       0.73      0.65      0.67      2000
weighted avg       0.83      0.85      0.83      2000
```

In [100]:

```python
# K Nearest Neighbour Algorithm
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(x_train_tfidf, y_train)

# Making predictions on the test set
predictions_knn = knn_classifier.predict(x_test_tfidf)


print("Confusion Matrix for k-Nearest Neighbors (KNN) Classifier:")
print(confusion_matrix(y_test, predictions_knn))

accuracy = accuracy_score(y_test, predictions_knn) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_knn))
```

```
Confusion Matrix for k-Nearest Neighbors (KNN) Classifier:
[[  40   51  122]
 [  60  200  105]
 [ 119  140 1163]]
Accuracy Score: 70.15%
Classification Report:
              precision    recall  f1-score   support

     average       0.18      0.19      0.19       213
         bad       0.51      0.55      0.53       365
        good       0.84      0.82      0.83      1422

    accuracy                           0.70      2000
   macro avg       0.51      0.52      0.51      2000
weighted avg       0.71      0.70      0.70      2000
```

In [105]:

```
# XGBoost Classifier
#!pip install xgboost
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)


# Creating and training the XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(
    learning_rate=0.1,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    n_estimators=100,
    random_state=101
)
xgb_classifier.fit(x_train_tfidf, y_train_encoded)


predictions_xgb = xgb_classifier.predict(x_test_tfidf)
predictions_xgb_labels = label_encoder.inverse_transform(predictions_xg

# Evaluating the XGBoost Classifier
print("Confusion Matrix for XGBoost Classifier:")
print(confusion_matrix(y_test, predictions_xgb_labels))

accuracy = accuracy_score(y_test, predictions_xgb_labels) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_xgb_labels))
```

```
Confusion Matrix for XGBoost Classifier:
[[  27   28  158]
 [  18  226  121]
 [  16   20 1386]]
Accuracy Score: 81.95%
Classification Report:
              precision    recall  f1-score   support

     average       0.44      0.13      0.20       213
         bad       0.82      0.62      0.71       365
        good       0.83      0.97      0.90      1422

    accuracy                           0.82      2000
   macro avg       0.70      0.57      0.60      2000
weighted avg       0.79      0.82      0.79      2000
```

In [107]:

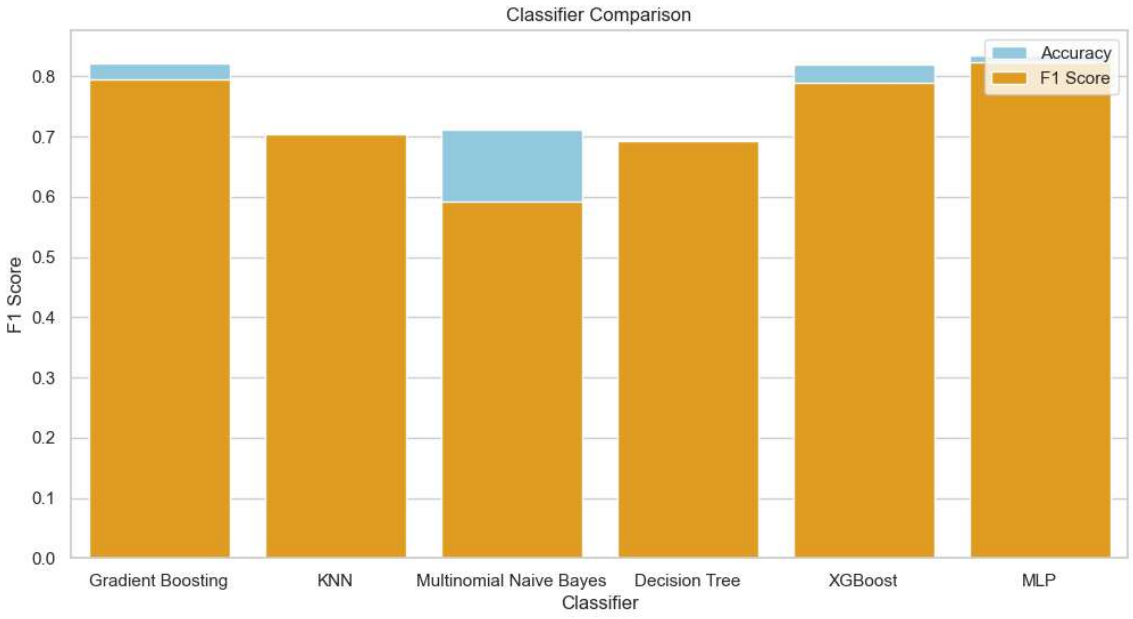
```python
# MULTILAYER PERCEPTRON CLASSIFIER
from sklearn.neural_network import MLPClassifier

mlp_classifier = MLPClassifier(
    hidden_layer_sizes=(100,),
    activation='relu',
    solver='adam',
    max_iter=200,
    random_state=101
)
mlp_classifier.fit(x_train_tfidf, y_train_encoded)

# Making predictions on the test set
predictions_mlp = mlp_classifier.predict(x_test_tfidf)

# Inverse transform the numerical labels back to original string labels
predictions_mlp_labels = label_encoder.inverse_transform(predictions_ml

# Evaluating the MLP Classifier
print("Confusion Matrix for MLP Classifier:")
print(confusion_matrix(y_test, predictions_mlp_labels))

accuracy = accuracy_score(y_test, predictions_mlp_labels) * 100
print("Accuracy Score: {:.2f}%".format(accuracy))

print("Classification Report:")
print(classification_report(y_test, predictions_mlp_labels))
```

```
Confusion Matrix for MLP Classifier:
[[  58   47  108]
 [  30  279   56]
 [  58   32 1332]]
Accuracy Score: 83.45%
Classification Report:
              precision    recall  f1-score   support

     average       0.40      0.27      0.32       213
         bad       0.78      0.76      0.77       365
        good       0.89      0.94      0.91      1422

    accuracy                           0.83      2000
   macro avg       0.69      0.66      0.67      2000
weighted avg       0.82      0.83      0.82      2000
```

In [112]:

```
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

#graph for results comparison
results = {'Classifier': [], 'Accuracy': [], 'F1 Score': []}

# Gradient Boosting Classifier
gbi = GradientBoostingClassifier(learning_rate=0.1, max_depth=5, max_fe
gbi.fit(x_train_tfidf, y_train)
pred_gbi = gbi.predict(x_test_tfidf)
accuracy_gbi = accuracy_score(y_test, pred_gbi)
f1_gbi = f1_score(y_test, pred_gbi, average='weighted')
results['Classifier'].append('Gradient Boosting')
results['Accuracy'].append(accuracy_gbi)
results['F1 Score'].append(f1_gbi)

# K-Nearest Neighbors (KNN) Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(x_train_tfidf, y_train_encoded)
pred_knn = knn_classifier.predict(x_test_tfidf)
accuracy_knn = accuracy_score(y_test_encoded, pred_knn)
f1_knn = f1_score(y_test_encoded, pred_knn, average='weighted')
results['Classifier'].append('KNN')
results['Accuracy'].append(accuracy_knn)
results['F1 Score'].append(f1_knn)

# Multinomial Naive Bayes Classifier
mnb_classifier = MultinomialNB()
mnb_classifier.fit(x_train_tfidf, y_train_encoded)
pred_mnb = mnb_classifier.predict(x_test_tfidf)
accuracy_mnb = accuracy_score(y_test_encoded, pred_mnb)
f1_mnb = f1_score(y_test_encoded, pred_mnb, average='weighted')
results['Classifier'].append('Multinomial Naive Bayes')
results['Accuracy'].append(accuracy_mnb)
results['F1 Score'].append(f1_mnb)

# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=101)
dt_classifier.fit(x_train_tfidf, y_train_encoded)
pred_dt = dt_classifier.predict(x_test_tfidf)
accuracy_dt = accuracy_score(y_test_encoded, pred_dt)
f1_dt = f1_score(y_test_encoded, pred_dt, average='weighted')
results['Classifier'].append('Decision Tree')
results['Accuracy'].append(accuracy_dt)
results['F1 Score'].append(f1_dt)

# XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(learning_rate=0.1, max_depth=5, subs
xgb_classifier.fit(x_train_tfidf, y_train_encoded)
pred_xgb = xgb_classifier.predict(x_test_tfidf)
accuracy_xgb = accuracy_score(y_test_encoded, pred_xgb)
f1_xgb = f1_score(y_test_encoded, pred_xgb, average='weighted')
results['Classifier'].append('XGBoost')
results['Accuracy'].append(accuracy_xgb)
```

```
results['F1 Score'].append(f1_xgb)

# Multi-layer Perceptron (MLP) Classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100,), activation='r
mlp_classifier.fit(x_train_tfidf, y_train_encoded)
pred_mlp = mlp_classifier.predict(x_test_tfidf)
accuracy_mlp = accuracy_score(y_test_encoded, pred_mlp)
f1_mlp = f1_score(y_test_encoded, pred_mlp, average='weighted')
results['Classifier'].append('MLP')
results['Accuracy'].append(accuracy_mlp)
results['F1 Score'].append(f1_mlp)

# Convert results to DataFrame for easy plotting
import pandas as pd
df_results = pd.DataFrame(results)

# Plotting the results
plt.figure(figsize=(12, 6))
sns.set(style="whitegrid")
sns.barplot(x='Classifier', y='Accuracy', data=df_results, color='skybl
sns.barplot(x='Classifier', y='F1 Score', data=df_results, color='orang
plt.title('Classifier Comparison')
plt.legend(loc='upper right')
plt.show()

```



In [ ]:

```
1
```