Feature Engineering

In [1]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Read the TSV file
file_path = "Restaurant_Reviews.tsv"
df = pd.read_csv(file_path, delimiter='\t', quoting=3)

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')

# Fit and transform the reviews to TF-IDF vectors
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Review'])

# Convert TF-IDF matrix to a dense array
tfidf_features = tfidf_matrix.toarray()

# Create a new DataFrame with TF-IDF features
tfidf_df = pd.DataFrame(tfidf_features, columns=tfidf_vectorizer.get_feature_names_

# Concatenate the TF-IDF DataFrame with the original DataFrame
final_df = pd.concat([df, tfidf_df], axis=1)

# Display the final DataFrame
print(final_df)
```

```
                                                Review  Liked   10  100   12  \
0                             Wow... Loved this place.      1  0.0  0.0  0.0   
1                                   Crust is not good.      0  0.0  0.0  0.0   
2            Not tasty and the texture was just nasty.      0  0.0  0.0  0.0   
3    Stopped by during the late May bank holiday of...      1  0.0  0.0  0.0   
4    The selection on the menu was great and so wer...      1  0.0  0.0  0.0   
..                                                 ...    ...  ...  ...  ...   
995  I think food should have flavor and texture an...      0  0.0  0.0  0.0   
996                           Appetite instantly gone.      0  0.0  0.0  0.0   
997  Overall I was not impressed and would not go b...      0  0.0  0.0  0.0   
998  The whole experience was underwhelming, and I ...      0  0.0  0.0  0.0   
999  Then, as if I hadn't wasted enough of my life ...      0  0.0  0.0  0.0   

      20   30   35   40  absolutely  ...  year  years  yellow  yellowtail  \
0    0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
1    0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
2    0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
3    0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
4    0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
..   ...  ...  ...  ...         ...  ...   ...    ...     ...         ...   
995  0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
996  0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
997  0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
998  0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   
999  0.0  0.0  0.0  0.0         0.0  ...   0.0    0.0     0.0         0.0   

     yelpers  yucky  yukon  yum  yummy  zero  
0        0.0    0.0    0.0  0.0    0.0   0.0  
1        0.0    0.0    0.0  0.0    0.0   0.0  
2        0.0    0.0    0.0  0.0    0.0   0.0  
3        0.0    0.0    0.0  0.0    0.0   0.0  
4        0.0    0.0    0.0  0.0    0.0   0.0  
..       ...    ...    ...  ...    ...   ...  
995      0.0    0.0    0.0  0.0    0.0   0.0  
996      0.0    0.0    0.0  0.0    0.0   0.0  
997      0.0    0.0    0.0  0.0    0.0   0.0  
998      0.0    0.0    0.0  0.0    0.0   0.0  
999      0.0    0.0    0.0  0.0    0.0   0.0  

[1000 rows x 1002 columns]
```

Training and Testing

In [2]:

```python
from sklearn.model_selection import train_test_split

# Split the data into features (X) and labels (y)
X = final_df.drop(['Review', 'Liked'], axis=1)  # Features excluding review text ar
y = final_df['Liked']  # Sentiment labels

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Logistic Regression, SVM, Naive Bayes, XGBoost, KNN

In [85]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matri
from tqdm import tqdm
```

```python
import numpy as np
import pandas as pd

import warnings

# Settings the warnings to be ignored
warnings.filterwarnings('ignore')

# Initialize models
models = [
    LogisticRegression(),
    MultinomialNB(),
    SVC(),
    GradientBoostingClassifier(),
    KNeighborsClassifier()
]

# Initialize an empty list to store results for all models
results_list = []

# Iterate through models
for model in models:
    # Model Training
    model.fit(X_train, y_train)

    # Model Evaluation with Progress Bar
    y_pred = []  # Initialize an empty list to store predictions

    # Use tqdm to create a progress bar for predicting
    with tqdm(total=len(X_test)) as pbar:
        for i in range(len(X_test)):
            y_pred.append(model.predict(X_test.iloc[i:i+1]))  # Predict for one ins
            pbar.update(1)

    # Convert the list of predictions to a numpy array
    y_pred = np.array(y_pred).flatten()

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Generate classification report and confusion matrix
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Save analysis results in a dictionary
    model_results = {
        'Model': str(model),
        'Accuracy': accuracy,
        'Classification Report': report,
        'Confusion Matrix': conf_matrix
    }

    # Append results to the list
    results_list.append(model_results)

# Convert the list of results dictionaries into a DataFrame
results_df = pd.DataFrame(results_list)
```

```
100%|██████████████████████████████████████████████████████████████████████████
██| 200/200 [00:01<00:00, 152.91it/s]
100%|██████████████████████████████████████████████████████████████████████████
██| 200/200 [00:01<00:00, 150.23it/s]
100%|██████████████████████████████████████████████████████████████████████████
██| 200/200 [00:01<00:00, 119.42it/s]
100%|██████████████████████████████████████████████████████████████████████████
██| 200/200 [00:01<00:00, 155.15it/s]
100%|██████████████████████████████████████████████████████████████████████████
██| 200/200 [00:01<00:00, 118.62it/s]
```

In [56]:

```python
from tabulate import tabulate

# Create a well-formatted table using tabulate
table = tabulate(results_df, headers='keys', tablefmt='grid')

# Print the table
print(table)
```

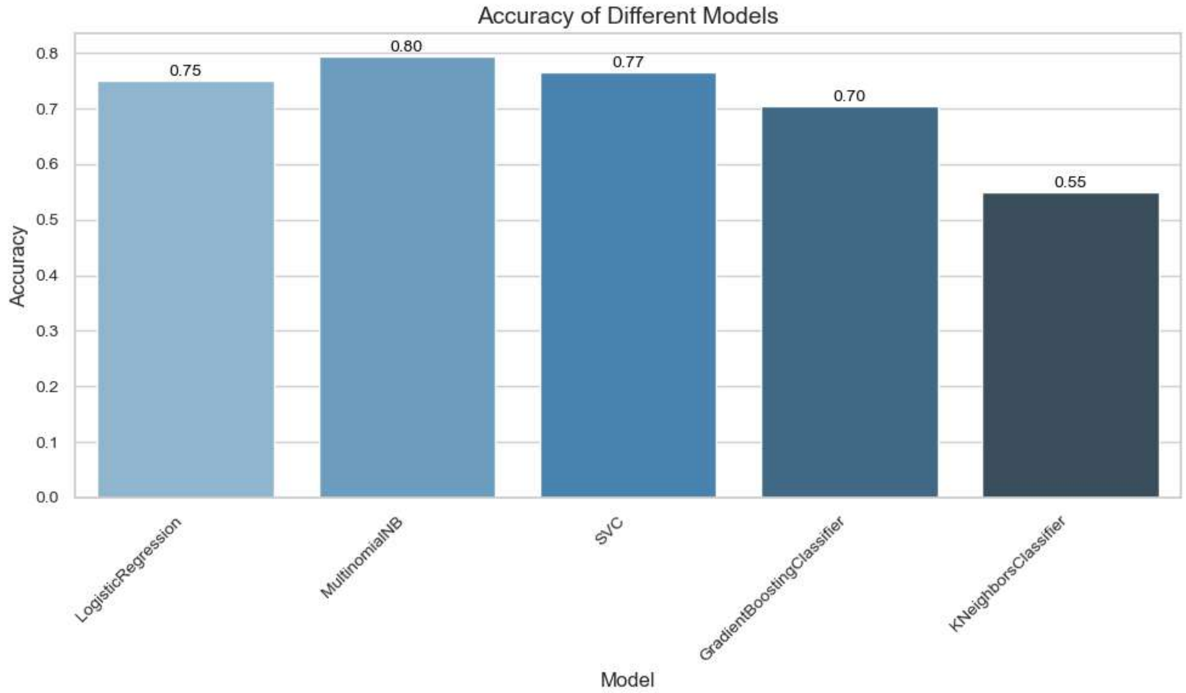
```
+----+-------------------------------+------------+-----------------------------------
-----------------------+--------------------+
|    | Model                         |   Accuracy | Classification Report
| Confusion Matrix   |
+====+===============================+============+===================================
=======================+====================+
|  0 | LogisticRegression()          |      0.75  | precision    recall  f1-score
support                | [[82 14]           |
|    |                               |            |
|  [36 68]]          |
|    |                               |            |            0       0.69      0.
85      0.77        96 |                    |
|    |                               |            |            1       0.83      0.
65      0.73       104 |                    |
|    |                               |            |
|                    |
|    |                               |            |     accuracy
0.75       200 |                    |
|    |                               |            |    macro avg       0.76      0.
75      0.75       200 |                    |
|    |                               |            | weighted avg       0.76      0.
75      0.75       200 |                    |
+----+-------------------------------+------------+-----------------------------------
-----------------------+--------------------+
|  1 | MultinomialNB()               |      0.795 | precision    recall  f1-score
support                | [[77 19]           |
|    |                               |            |
|  [22 82]]          |
|    |                               |            |            0       0.78      0.
80      0.79        96 |                    |
|    |                               |            |            1       0.81      0.
79      0.80       104 |                    |
|    |                               |            |
|                    |
|    |                               |            |     accuracy
0.80       200 |                    |
|    |                               |            |    macro avg       0.79      0.
80      0.79       200 |                    |
|    |                               |            | weighted avg       0.80      0.
80      0.80       200 |                    |
+----+-------------------------------+------------+-----------------------------------
-----------------------+--------------------+
|  2 | SVC()                         |      0.765 | precision    recall  f1-score
support                | [[81 15]           |
|    |                               |            |
|  [32 72]]          |
|    |                               |            |            0       0.72      0.
84      0.78        96 |                    |
|    |                               |            |            1       0.83      0.
69      0.75       104 |                    |
|    |                               |            |
|                    |
|    |                               |            |     accuracy
0.77       200 |                    |
|    |                               |            |    macro avg       0.77      0.
77      0.76       200 |                    |
|    |                               |            | weighted avg       0.77      0.
77      0.76       200 |                    |
+----+-------------------------------+------------+-----------------------------------
-----------------------+--------------------+
|  3 | GradientBoostingClassifier()  |      0.705 | precision    recall  f1-score
support                | [[88  8]           |
|    |                               |            |
|  [51 53]]          |
```

```
|    |                              |            |            0       0.63      0.
92       0.75        96 |                    |
|    |                              |            |            1       0.87      0.
51       0.64       104 |                    |
|    |                              |            |
|                      |
|    |                              |            |     accuracy
0.70        200 |                   |
|    |                              |            |    macro avg       0.75      0.
71       0.70       200 |                    |
|    |                              |            | weighted avg       0.76      0.
70       0.69       200 |                    |
+----+------------------------------+------------+-------------------------------
----------------------+--------------------+
|  4 | KNeighborsClassifier()       |       0.55 | precision    recall  f1-score
support                   | [[95  1]           |
|    |                              |            |
|   [89 15]]            |
|    |                              |            |            0       0.52      0.
99       0.68        96 |                    |
|    |                              |            |            1       0.94      0.
14       0.25       104 |                    |
|    |                              |            |
|                      |
|    |                              |            |     accuracy
0.55        200 |                   |
|    |                              |            |    macro avg       0.73      0.
57       0.46       200 |                    |
|    |                              |            | weighted avg       0.74      0.
55       0.46       200 |                    |
+----+------------------------------+------------+-------------------------------
----------------------+--------------------+
```

In [57]:

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

# Remove parentheses from 'Model' column
results_df['Model'] = results_df['Model'].str.replace(r'\([^)]*\)', '')

# Set the style using seaborn
sns.set(style="whitegrid")

# Plot the accuracy of each model
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='Model', y='Accuracy', data=results_df, palette='Blues_d')
plt.xlabel('Model', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Accuracy of Different Models', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

# Annotate values on each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_he

plt.tight_layout()

# Display the plot
plt.show()
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_11852\1495200454.py:5: FutureWarning: T
he default value of regex will change from True to False in a future version.
  results_df['Model'] = results_df['Model'].str.replace(r'\([^)]*\)', '')
```



In [21]:

```
# Find the model with the highest accuracy
best_accuracy_model = results_df.loc[results_df['Accuracy'].idxmax()]

# Print the best accuracy model's details
print("\nBest Accuracy Model:")
print(best_accuracy_model)
```

```
Best Accuracy Model:
Model                                                          MultinomialNB()
Accuracy                                                                 0.795
Classification Report                            precision    recall  f1-score   ...
Confusion Matrix                                          [[77, 19], [22, 82]]
Name: 1, dtype: object
```

In [75]:

```
# Extract the "Model" and "Classification Report" columns from results_df
selected_columns = results_df[["Model", "Classification Report"]]

# Iterate through rows and extract the second f1-score value for each model
for index, row in selected_columns.iterrows():
    model = row["Model"]
    classification_report = row["Classification Report"]

    # Split the classification report into lines
    report_lines = classification_report.split('\n')

    # Find the line containing "macro avg" (assumes the f1-score is in this line)
    f1_score_line = None
    for line in report_lines:
        if "macro avg" in line:
            f1_score_line = line
            break

    if f1_score_line:
        f1_scores = [float(score) for score in f1_score_line.split() if score != "
        if len(f1_scores) >= 2:
            f1_score_second_value = f1_scores[1]  # Extract the second value from 
            print(f"Model: {model}, f1-score: {f1_score_second_value:.2f}")
        else:
            print(f"Model: {model}, f1-score not found")
    else:
        print(f"Model: {model}, f1-scores not found")
```

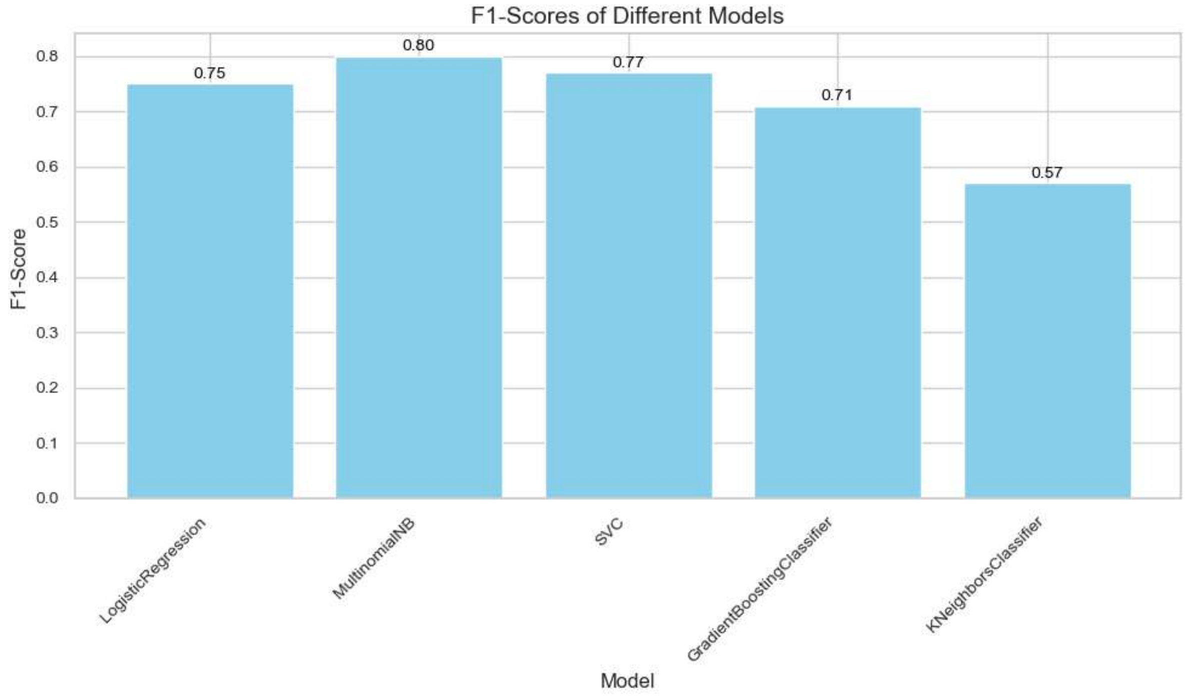
```
Model: LogisticRegression, f1-score: 0.75
Model: MultinomialNB, f1-score: 0.80
Model: SVC, f1-score: 0.77
Model: GradientBoostingClassifier, f1-score: 0.71
Model: KNeighborsClassifier, f1-score: 0.57
```

In [81]:

```python
import matplotlib.pyplot as plt

# Extract the "Model" and "Classification Report" columns from results_df
selected_columns = results_df[["Model", "Classification Report"]]

# Initialize lists to store model names and f1-scores
model_names = []
f1_scores = []

# Iterate through rows and extract the second f1-score value for each model
for index, row in selected_columns.iterrows():
    model = row["Model"]
    classification_report = row["Classification Report"]

    # Split the classification report into lines
    report_lines = classification_report.split('\n')

    # Find the line containing "macro avg" (assumes the f1-score is in this line)
    f1_score_line = None
    for line in report_lines:
        if "macro avg" in line:
            f1_score_line = line
            break

    if f1_score_line:
        f1_scores_list = [float(score) for score in f1_score_line.split() if score
        if len(f1_scores_list) >= 2:
            f1_score_second_value = f1_scores_list[1]  # Extract the second value
            model_names.append(model)
            f1_scores.append(f1_score_second_value)
        else:
            print(f"Model: {model}, f1-score not found")
    else:
        print(f"Model: {model}, f1-scores not found")
```

In [82]:

```python
# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(model_names, f1_scores, color='skyblue')
plt.xlabel('Model', fontsize=12)
plt.ylabel('F1-Score', fontsize=12)
plt.title('F1-Scores of Different Models', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

# Annotate values on each bar
for i, v in enumerate(f1_scores):
    plt.text(i, v + 0.01, f'{v:.2f}', ha='center', fontsize=10, color='black')

plt.tight_layout()

# Show the plot
plt.show()
```



In [83]:

```
# Find the index of the highest f1-score
max_f1_index = f1_scores.index(max(f1_scores))
highest_f1_score = max(f1_scores)
highest_f1_model = model_names[max_f1_index]

# Print the highest f1-score and its corresponding model
print(f"Highest F1-Score: {highest_f1_score:.2f} for Model: {highest_f1_model}")
```

```
Highest F1-Score: 0.80 for Model: MultinomialNB
```