Sentiment Classification of Restaurant Reviews Using Machine Learning Techniques

Group 8 ML1

Import required libraries

In [2]:
```
# Data handling
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Text preprocessing
import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer

# Dimensionality reduction
from sklearn.decomposition import PCA

# Data preprocessing / model prep
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Machine learning models
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier

# Evaluation / metrics
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, classificat
import math
```

In [3]:
```
# Load data
df = pd.read_csv("restaurant_reviews_sample.csv")
df.head(5)
```

Out[3]:

	review_id	user_id	business_id	stars	
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	de e a
1	saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3.0	I as
2	AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5.0	Y di de
3	Sx8TMOWLNuJBWer-0pcmoA	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	4.0	ov
4	JrIxlS1TzJ-iCu79ul40cQ	eUta8W_HdHMXPzLBBZhL1A	04UD14gamNjLY0IDYVhHJg	1.0	fı cu

Data prepreparation

In [4]:

```
# Keep required columns
df = df[['review_id', 'text', 'stars']].copy()

# Drop missing
df = df.dropna(subset=['text', 'stars'])

# Remove duplicate rows & duplicate IDs
df = df.drop_duplicates().drop_duplicates(subset='review_id')

# Rename the column 'text' to 'reviews'
df = df.rename(columns={'text': 'reviews'})
```

```
# Create sentiment label
def label_sentiment(star):
    if star >= 4:
        return 1
    elif star <= 2:
        return 0
    else:
        return None

df['sentiment'] = df['stars'].apply(label_sentiment)

# Drop neutral reviews
df = df.dropna(subset=['sentiment'])

# Keep only needed columns for modeling
reviews_df = df[['reviews', 'sentiment']].copy()

reviews_df.head()
```

Out[4]:

	reviews	sentiment
2	Wow! Yummy, different, delicious. Our favo...	1.0
3	Cute interior and owner (?) gave us tour of up...	1.0
4	I am a long term frequent customer of this est...	0.0
5	Amazingly amazing wings and homemade bleu chee...	1.0
7	Locals recommended Milktooth, and it's an amaz...	1.0

EDA

In [5]:
```
df = reviews_df.copy()

# Dataset Overview
df.head()
df.info()
df.describe(include='object')
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 43720 entries, 2 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   reviews    43720 non-null  object 
 1   sentiment  43720 non-null  float64
dtypes: float64(1), object(1)
memory usage: 1.0+ MB
```

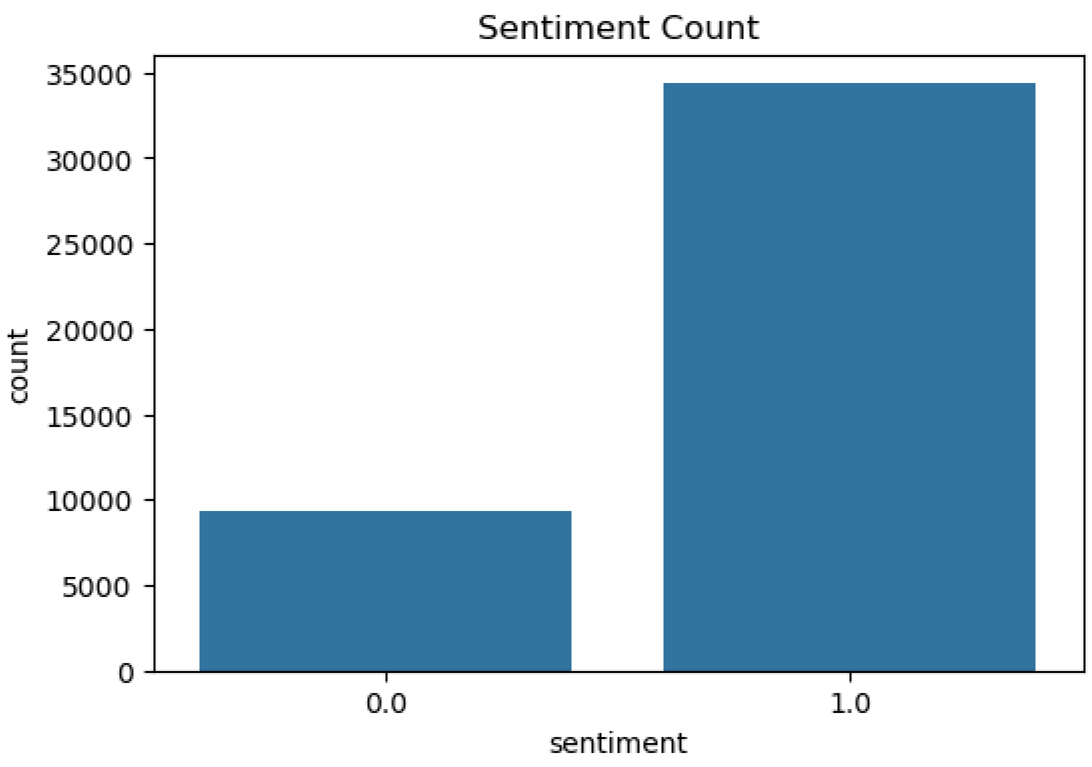
Out[5]:
```
reviews      0
sentiment    0
dtype: int64
```

In [6]:
```
#Counts positives and negatives
df['sentiment'].value_counts()
```

Out[6]:
```
sentiment
1.0    34335
0.0     9385
Name: count, dtype: int64
```

Sentiment distribution

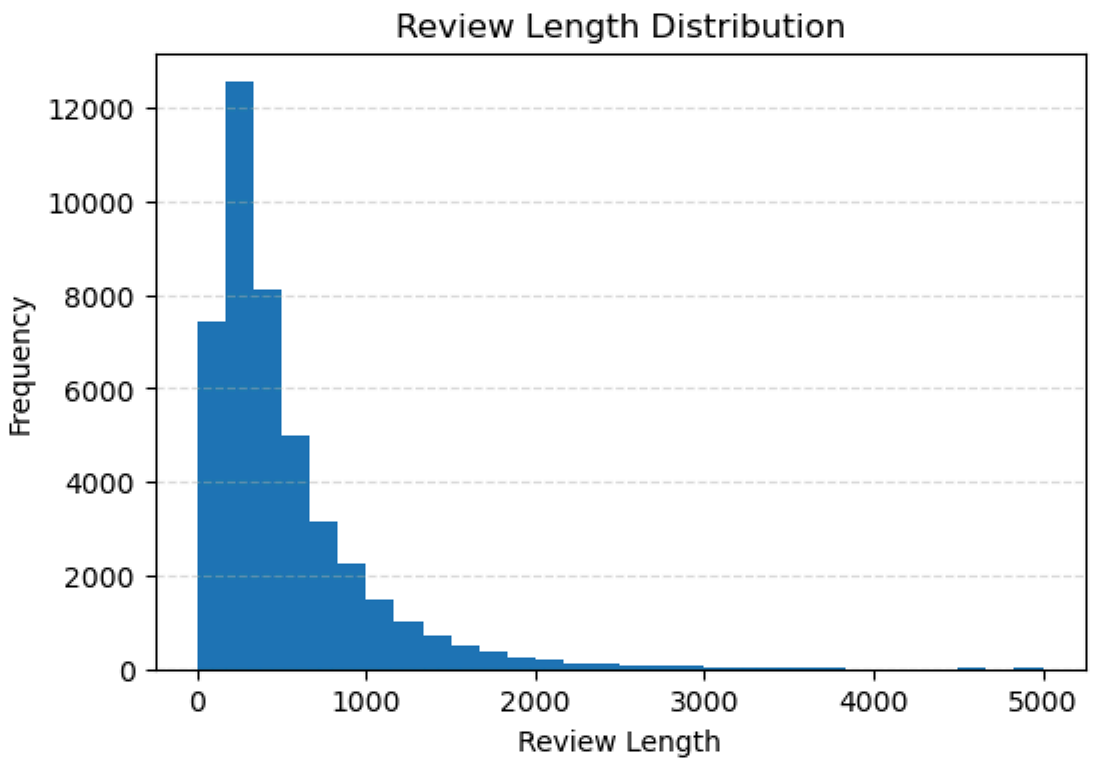
In [7]:
```
#Sentiment distribution countplot
plt.figure(figsize=(6,4))
sns.countplot(x='sentiment', data=df)
plt.title("Sentiment Count")
plt.show()
```



Review length distribution

In [8]:
```
#Creating review_length column
df['review_length'] = df['reviews'].astype(str).apply(len)
```

In [9]:
```
#Histogram of Review Length (distribution)
plt.figure(figsize=(6,4))
plt.hist(df['review_length'], bins=30)
plt.xlabel("Review Length")
plt.ylabel("Frequency")
plt.title("Review Length Distribution")
plt.grid(axis='y', linestyle='--', alpha=0.4)
plt.show()
```



Basic statistics (mean, median, mode, etc.)

In [10]:
```
#Summary Statistics (Mean, Median, Mode, Min, Max, SD)
print("Mean:", df['review_length'].mean())
print("Median:", df['review_length'].median())
print("Mode:", df['review_length'].mode()[0])
print("Std Dev:", df['review_length'].std())
print("Min:", df['review_length'].min())
print("Max:", df['review_length'].max())
```

```
Mean: 522.4263037511437
Median: 369.0
Mode: 233
Std Dev: 486.472346504319
Min: 3
Max: 5000
```

Text Pre-Processing

In [11]:
```
# Text Cleaning
import re
import string
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

def clean_text(text):
    text = text.lower()  # lowercase
    text = re.sub(f"[{string.punctuation}]", "", text)  # remove punctuation
    words = text.split()
    words = [w for w in words if w not in ENGLISH_STOP_WORDS]  # remove stopwords
    return " ".join(words)
```

```
reviews_df['clean_reviews'] = reviews_df['reviews'].apply(clean_text)
```

Train/Test Split + TF-IDF Vectorization

In [12]:
```
from sklearn.model_selection import train_test_split

# Define features and target
X = reviews_df['clean_reviews']  # cleaned text column
y = reviews_df['sentiment']      # target column (0/1)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1, stratify=y
)

# Check
print("Train size:", X_train.shape[0])
print("Test size:", X_test.shape[0])
```

```
Train size: 34976
Test size: 8744
```

In [14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF Vectorization
tfidf = TfidfVectorizer(
    max_features=5000,       # limit features
    stop_words='english',    # remove stopwords (optional, already cleaned)
    ngram_range=(1,2)        # unigrams + bigrams
)

# Fit TF-IDF only on training data
X_train_tfidf = tfidf.fit_transform(X_train)

# Transform test data using the same TF-IDF
X_test_tfidf = tfidf.transform(X_test)

print("TF-IDF Train:", X_train_tfidf.shape)
print("TF-IDF Test :", X_test_tfidf.shape)
```

```
TF-IDF Train: (34976, 5000)
TF-IDF Test : (8744, 5000)
```

In [15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)
import pandas as pd
```

PCA + Scaling

In [17]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def apply_pca_and_scale(X_train, X_test, n=100):
    pca = PCA(n_components=n, random_state=1)
    X_train_pca = pca.fit_transform(X_train.toarray())
    X_test_pca = pca.transform(X_test.toarray())

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_pca)
    X_test_scaled = scaler.transform(X_test_pca)

    return X_train_scaled, X_test_scaled

X_train_lr, X_test_lr = apply_pca_and_scale(X_train_tfidf, X_test_tfidf)
X_train_knn, X_test_knn = apply_pca_and_scale(X_train_tfidf, X_test_tfidf)
X_train_mlr, X_test_mlr = apply_pca_and_scale(X_train_tfidf, X_test_tfidf)
```

In [21]:
```
def print_classification_metrics(y_true, y_pred, model_name="Model"):
    cm = confusion_matrix(y_true, y_pred)
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"\n==================== {model_name} ====================")
    print("Confusion Matrix (rows = true, cols = predicted):")
    print(cm)
    print(f"\nAccuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")
    print("\nDetailed classification report:")
    print(classification_report(y_true, y_pred, digits=4))
```

Models, Logistice regression

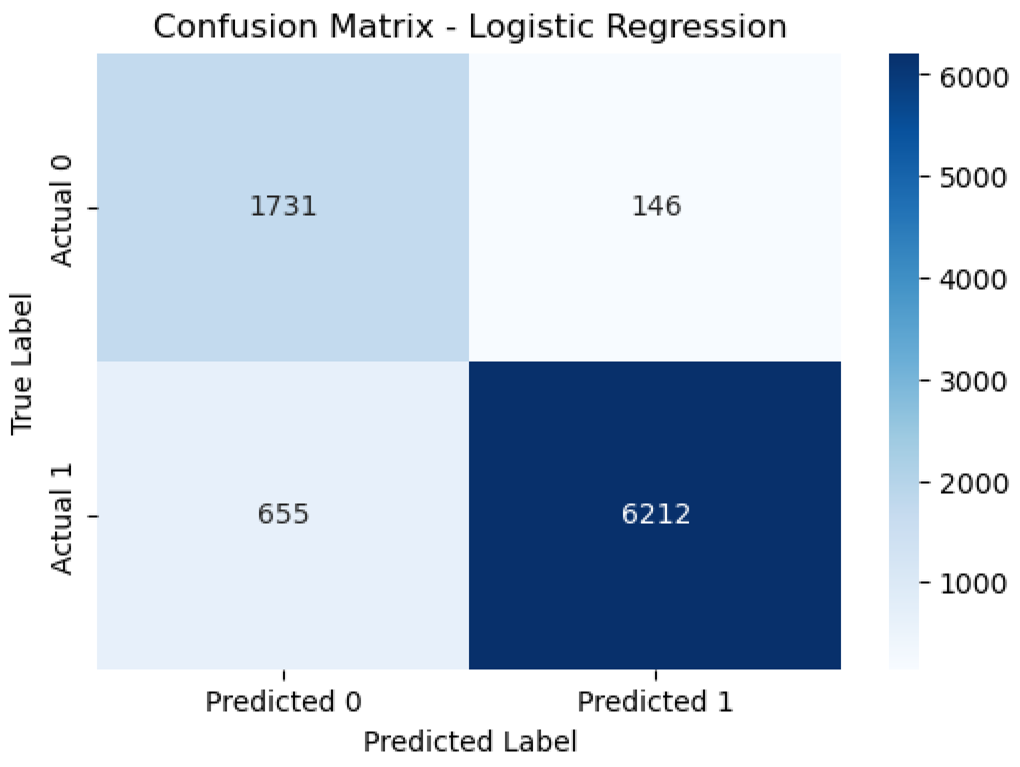
In [24]:
```
from sklearn.linear_model import LogisticRegression

# Initialize model with class balancing
log_reg = LogisticRegression(
    solver='lbfgs',
    max_iter=1000,
    random_state=1,
    n_jobs=-1,
    class_weight='balanced'
)

log_reg.fit(X_train_lr, y_train)

# Predictions on test set
```

```
y_pred_lr = log_reg.predict(X_test_lr)

# Metrics for Logistic Regression
print_classification_metrics(y_test, y_pred_lr,
                             model_name="Logistic Regression")
```

```
==================== Logistic Regression ====================
Confusion Matrix (rows = true, cols = predicted):
[[1731  146]
 [ 655 6212]]

Accuracy : 0.9084
Precision: 0.9770
Recall   : 0.9046
F1-score : 0.9394

Detailed classification report:
              precision    recall  f1-score   support

         0.0     0.7255    0.9222    0.8121      1877
         1.0     0.9770    0.9046    0.9394      6867

    accuracy                         0.9084      8744
   macro avg     0.8513    0.9134    0.8758      8744
weighted avg     0.9230    0.9084    0.9121      8744
```

In [27]:
```
y_pred_lr = log_reg.predict(X_test_lr)
acc_lr = accuracy_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
```

In [28]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_lr)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix - Logistic Regression")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()
```



KNN Classification model + metrics

In [32]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Try different k values and record accuracy
results = []
for k in range(1, 16):
    knn_tmp = KNeighborsClassifier(n_neighbors=k, weights='distance', n_jobs=-1)
    knn_tmp.fit(X_train_knn, y_train)
    y_pred_tmp = knn_tmp.predict(X_test_knn)
    acc_tmp = accuracy_score(y_test, y_pred_tmp)
    results.append({"k": k, "accuracy": acc_tmp})

results_df = pd.DataFrame(results)
print("KNN accuracy for different k values:")
print(results_df)

# Choose the k with highest accuracy on the test set
best_row = results_df.loc[results_df["accuracy"].idxmax()]
best_k = int(best_row["k"])
print(f"\nBest k based on test accuracy: k = {best_k}, "
      f"accuracy = {best_row['accuracy']:.4f}")

# Train final KNN model with best k
knn_best = KNeighborsClassifier(n_neighbors=best_k, weights='distance', n_jobs=-1)
knn_best.fit(X_train_knn, y_train)

# Predictions and metrics
y_pred_knn = knn_best.predict(X_test_knn)
```

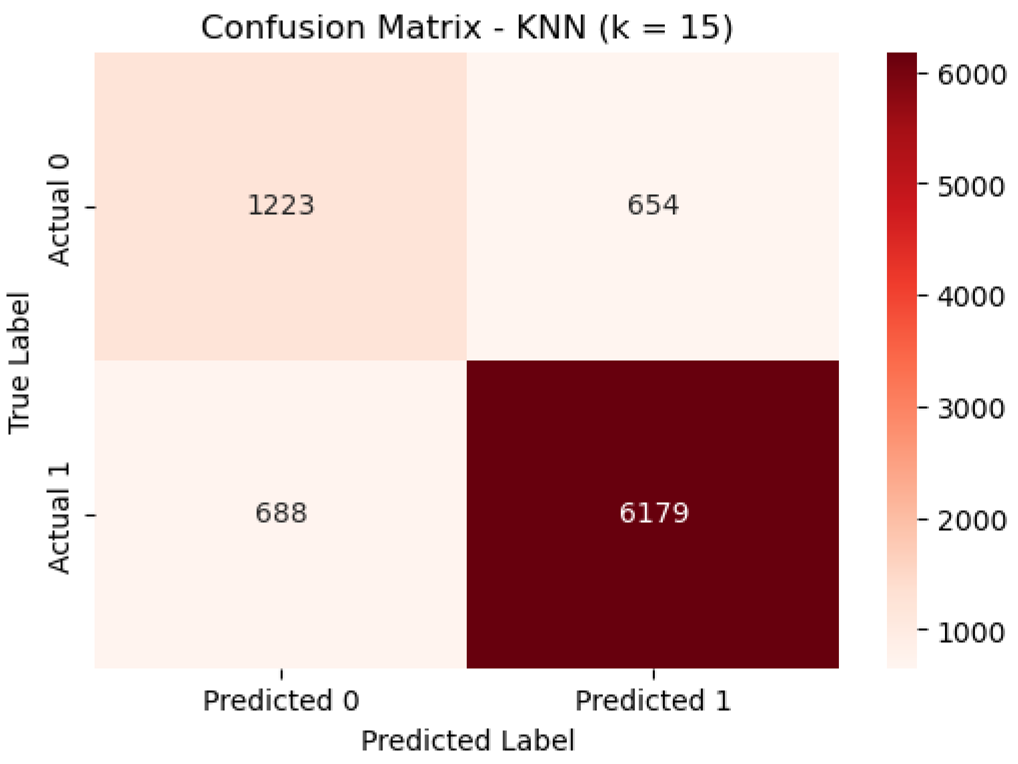
```
print_classification_metrics(y_test, y_pred_knn,
                             model_name=f'KNN (k = {best_k}, PCA + scaled)')
```

```
KNN accuracy for different k values:
     k  accuracy
0    1  0.743367
1    2  0.743481
2    3  0.793573
3    4  0.792086
4    5  0.810499
5    6  0.816560
6    7  0.824909
7    8  0.825595
8    9  0.827996
9   10  0.832113
10  11  0.836574
11  12  0.838289
12  13  0.842521
13  14  0.844122
14  15  0.846523

Best k based on test accuracy: k = 15, accuracy = 0.8465

==================== KNN (k = 15, PCA + scaled) ====================
Confusion Matrix (rows = true, cols = predicted):
[[1223  654]
 [ 688 6179]]

Accuracy : 0.8465
Precision: 0.9043
Recall   : 0.8998
F1-score : 0.9020

Detailed classification report:
              precision    recall  f1-score   support

         0.0     0.6400    0.6516    0.6457      1877
         1.0     0.9043    0.8998    0.9020      6867

    accuracy                         0.8465      8744
   macro avg     0.7721    0.7757    0.7739      8744
weighted avg     0.8476    0.8465    0.8470      8744
```

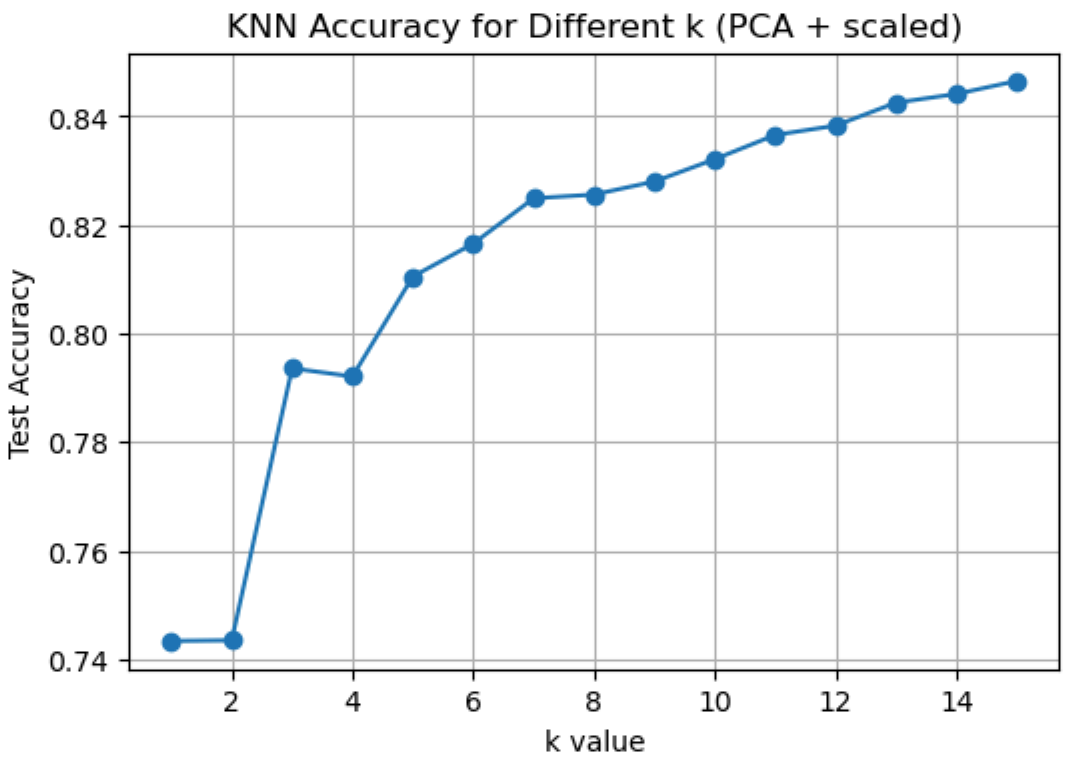
In [33]:
```
y_pred_knn = knn_best.predict(X_test_knn)
acc_knn = accuracy_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
```

In [34]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Confusion matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)

plt.figure(figsize=(6,4))
```

```
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Reds',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title(f"Confusion Matrix - KNN (k = {best_k})")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()
```



In [35]:
```
# Plot accuracy vs k
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
plt.plot(results_df['k'], results_df['accuracy'], marker='o')
plt.xlabel("k value")
plt.ylabel("Test Accuracy")
plt.title("KNN Accuracy for Different k (PCA + scaled)")
plt.grid(True)
plt.show()
```



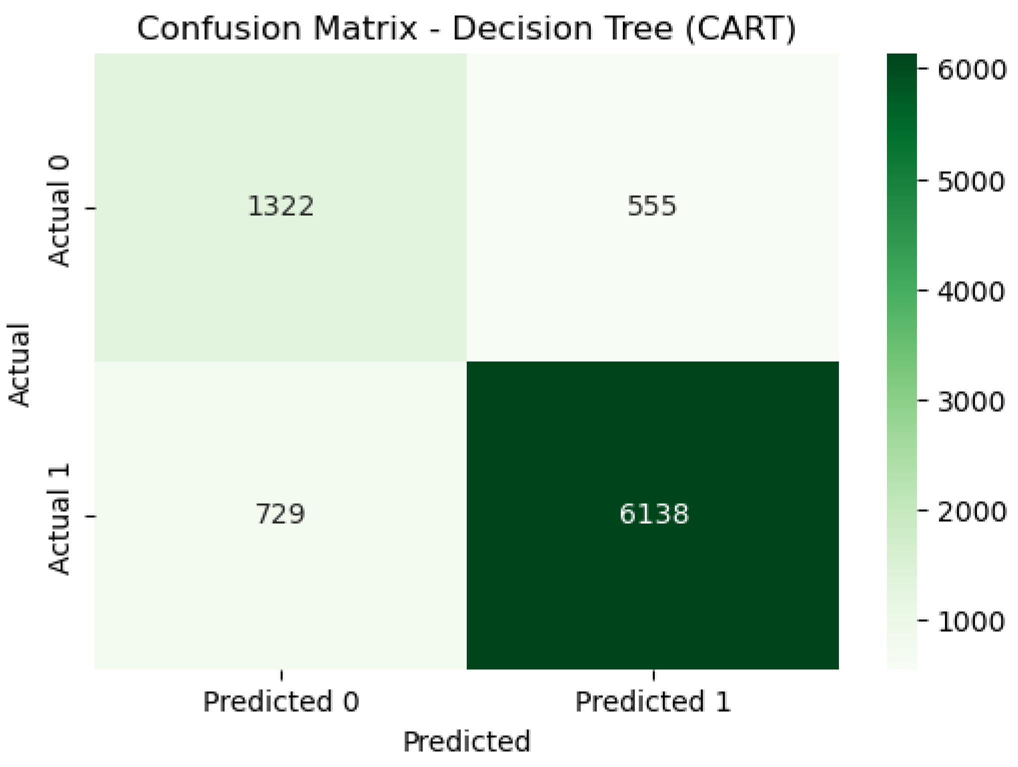
CART Decision Tree

In [36]:
```
from sklearn.tree import DecisionTreeClassifier

cart = DecisionTreeClassifier(
    class_weight='balanced',
    random_state=1)

# Train
cart.fit(X_train_tfidf, y_train)

# Predictions
y_pred_cart = cart.predict(X_test_tfidf)

# Metrics
print_classification_metrics(y_test, y_pred_cart,
                             model_name="Decision Tree (CART)")
```

```
==================== Decision Tree (CART) ====================
Confusion Matrix (rows = true, cols = predicted):
[[1322  555]
 [ 729 6138]]

Accuracy : 0.8532
Precision: 0.9171
Recall   : 0.8938
F1-score : 0.9053

Detailed classification report:
              precision    recall  f1-score   support

         0.0     0.6446    0.7043    0.6731      1877
         1.0     0.9171    0.8938    0.9053      6867

    accuracy                         0.8532      8744
   macro avg     0.7808    0.7991    0.7892      8744
weighted avg     0.8586    0.8532    0.8555      8744
```

In [37]:
```python
y_pred_cart = cart.predict(X_test_tfidf)
acc_cart = accuracy_score(y_test, y_pred_cart)
f1_cart = f1_score(y_test, y_pred_cart)
```

In [38]:
```python
cm_cart = confusion_matrix(y_test, y_pred_cart)

plt.figure(figsize=(6,4))
sns.heatmap(cm_cart, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix - Decision Tree (CART)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



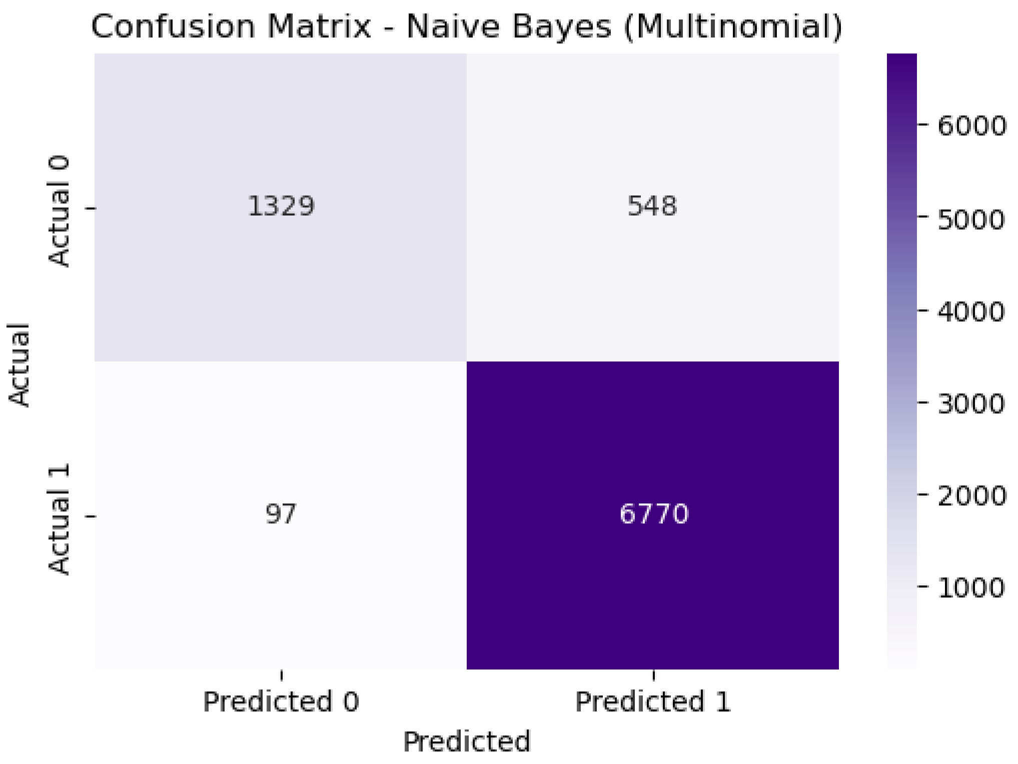
Naive Bayes

In [39]:
```
from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()

# Train NB on TF-IDF sparse matrix (correct for text)
nb_model.fit(X_train_tfidf, y_train)

# Predictions
nb_pred = nb_model.predict(X_test_tfidf)

# Use YOUR metric function here
print_classification_metrics(y_test, nb_pred,
                             model_name="Naive Bayes (Multinomial)")
```

```
==================== Naive Bayes (Multinomial) ====================
Confusion Matrix (rows = true, cols = predicted):
[[1329  548]
 [  97 6770]]

Accuracy : 0.9262
Precision: 0.9251
Recall   : 0.9859
F1-score : 0.9545

Detailed classification report:
              precision    recall  f1-score   support

         0.0     0.9320    0.7080    0.8047      1877
         1.0     0.9251    0.9859    0.9545      6867

    accuracy                         0.9262      8744
   macro avg     0.9285    0.8470    0.8796      8744
weighted avg     0.9266    0.9262    0.9224      8744
```

In [40]:
```python
y_pred_nb = nb_model.predict(X_test_tfidf)
acc_nb = accuracy_score(y_test, y_pred_nb)
f1_nb = f1_score(y_test, y_pred_nb)
```

In [41]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm_nb = confusion_matrix(y_test, nb_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Purples',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix - Naive Bayes (Multinomial)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



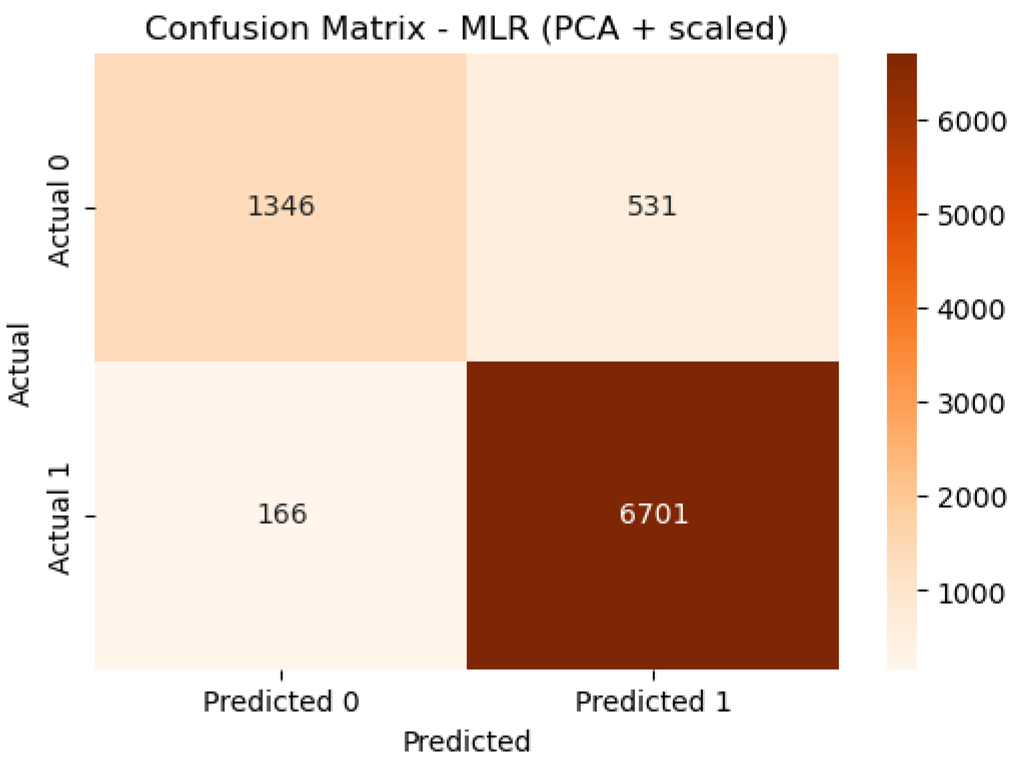
Multiple Linear Regression

In [42]:
```
from sklearn.linear_model import LinearRegression

mlr_model = LinearRegression()
mlr_model.fit(X_train_mlr, y_train)

# Predict continuous values (0–1)
mlr_pred_continuous = mlr_model.predict(X_test_mlr)

# Convert to binary
mlr_pred = (mlr_pred_continuous >= 0.5).astype(int)

# Use your metric function
print_classification_metrics(
    y_test, mlr_pred,
    model_name="Multiple Linear Regression (PCA + scaled)"
)
```

```
==================== Multiple Linear Regression (PCA + scaled) ====================
Confusion Matrix (rows = true, cols = predicted):
[[1346  531]
 [ 166 6701]]

Accuracy : 0.9203
Precision: 0.9266
Recall   : 0.9758
F1-score : 0.9506

Detailed classification report:
              precision    recall  f1-score   support

         0.0     0.8902    0.7171    0.7943      1877
         1.0     0.9266    0.9758    0.9506      6867

    accuracy                         0.9203      8744
   macro avg     0.9084    0.8465    0.8724      8744
weighted avg     0.9188    0.9203    0.9170      8744
```

In [43]:
```
y_pred_mlr = mlr_pred   # already computed
acc_mlr = accuracy_score(y_test, y_pred_mlr)
f1_mlr = f1_score(y_test, y_pred_mlr)
```

In [44]:
```
cm_mlr = confusion_matrix(y_test, mlr_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm_mlr, annot=True, fmt='d', cmap='Oranges',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix - MLR (PCA + scaled)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



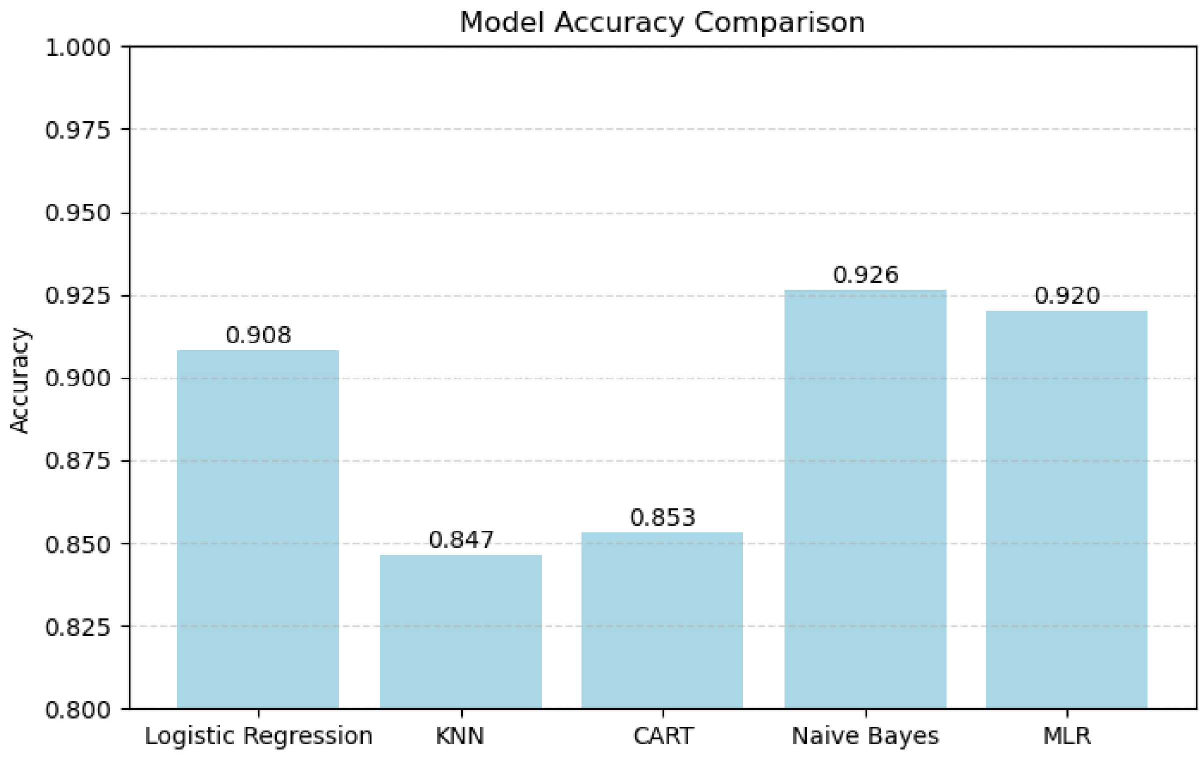
Results Comparison

In [86]:
```python
import pandas as pd

comparison_df = pd.DataFrame({
    "Model": ["Logistic Regression", "KNN", "CART", "Naive Bayes", "MLR"],
    "Accuracy": [acc_lr, acc_knn, acc_cart, acc_nb, acc_mlr],
    "F1-score": [f1_lr, f1_knn, f1_cart, f1_nb, f1_mlr]
})

comparison_df
```

Out[86]:

	Model	Accuracy	F1-score
0	Logistic Regression	0.908394	0.939433
1	KNN	0.846523	0.902044
2	CART	0.853156	0.905310
3	Naive Bayes	0.926235	0.954529
4	MLR	0.920288	0.950564

In [46]:
```python
plt.figure(figsize=(8,5))
plt.bar(comparison_df["Model"], comparison_df["Accuracy"], color='lightblue')
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.0)
plt.grid(axis='y', linestyle='--', alpha=0.5)
for i, v in enumerate(comparison_df["Accuracy"]):
    plt.text(i, v + 0.002, f"{v:.3f}", ha='center')

plt.show()
```


Model Accuracy Comparison
Accuracy
Logistic Regression
KNN
CART
Naive Bayes
MLR