1 数据集描述（包括统计数据、图表等）

2 读取数据

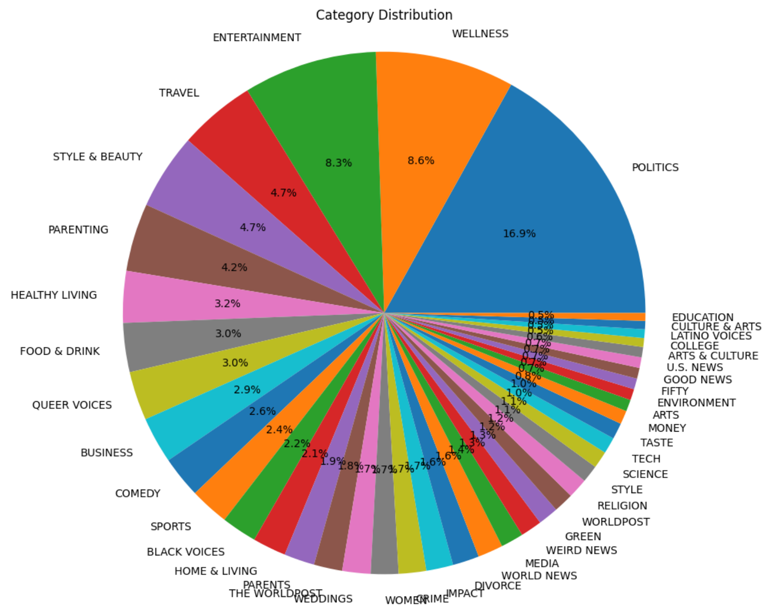
```
# 读取JSON文件并合并为列表
data = []
with open('News_Category.json', 'r') as file:
    for line in file:
        data.append(json.loads(line))

# 将JSON数据转换为DataFrame
df = pd.DataFrame(data)
```

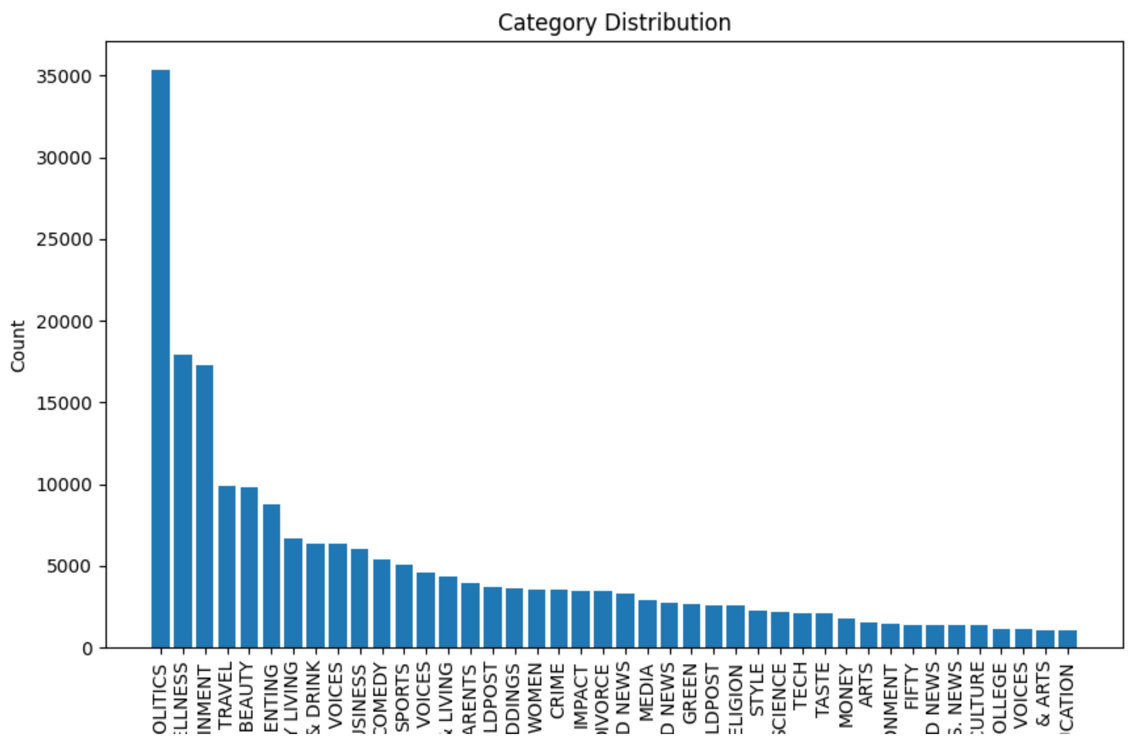
3 数据集描述（包括统计数据、图表等）

3.1 类别数目及占比统计

```
category_counts = df['category'].value_counts()
    # 绘制饼图
    plt.figure(figsize=(8, 6))
    plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%')
    plt.title('Category Distribution')
    plt.axis('equal')

    # 显示图形
    plt.show()

# 绘制柱状图
    plt.figure(figsize=(10, 6))
    plt.bar(category_counts.index, category_counts)
    plt.xlabel('Category')
    plt.ylabel('Count')
    plt.title('Category Distribution')
```

```
# 自动调整x轴标签的旋转角度
plt.xticks(rotation=90)

# 显示图形
plt.show()
```

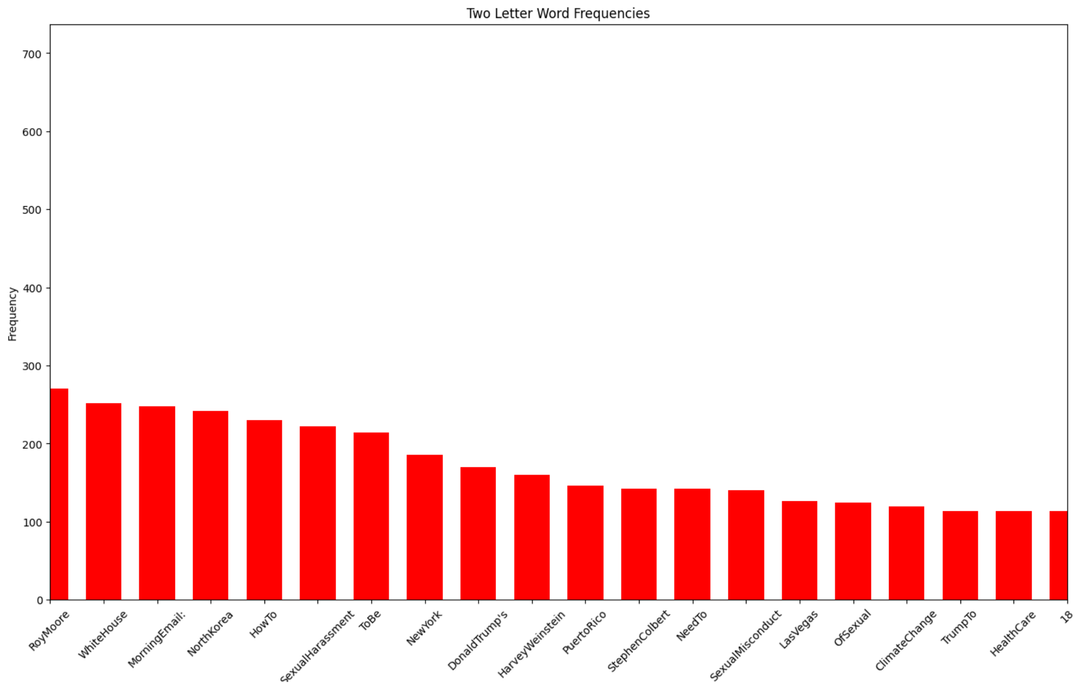




3.2 新闻词频统计

```
def most_common_words():

    nltk.download('stopwords')

    stop_words = set(stopwords.words('english'))
    tokenizer = nltk.RegexpTokenizer(r'\w+')

    content = ''.join(df['short_description']) + ''.join(df['headline'])
    tokens = tokenizer.tokenize(content.lower())
    filtered_tokens = [token for token in tokens if token not in stop_words]

    word_count = Counter(filtered_tokens)
    top_100 = word_count.most_common(100)
```

```
# 创建词云对象并生成词云图
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_fre

# 显示词云图
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```


Two Letter Word Frequencies
Frequency
0
100
200
300
400
500
600
700
RoyMoore
WhiteHouse
MorningEmail:
NorthKorea
HowTo
SexualHarassment
ToBe
NewYork
DonaldTrump's
HarveyWeinstein
PuertoRico
StephenColbert
NeedTo
SexualMisconduct
LasVegas
OfSexual
ClimateChange
TrumpTo
HealthCare
18

4 数据处理的流程

4.1 合并 headline 和 short-description 作为模型的输入数据

```
df['text'] = df.headline + " " + df.short_description
```

4.2 对文本进行清洗，包括将文本转换为小写、去除特殊字符等操作

```
def clean_str(string):
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    return string.lower()

df['text'] = df['text'].apply(clean_str)
```

4.3 将文本转化为 TF-IDF 特征向量

```
import re, string
re_tok = re.compile(f'([{string.punctuation}“” ¨«»®´·º½¾¿¡§£₤ ‘’ ])')
def tokenize(s): return re_tok.sub(r' \1 ', s).split()

tfidf_converter = TfidfVectorizer(min_df=3, max_df=0.9, strip_accents='unicode',tokenizer=tok
X_train_tfidf = tfidf_converter.fit_transform(X_train)
X_test_tfidf = tfidf_converter.transform(X_test)
```

4.4 划分训练集和测试集

```
X_train, X_test, y_train, y_test = train_test_split(df['fulltext_processed'], df['category'],
```

5 分类模型选择及设计、训练、验证、测试

5.1 朴素贝叶斯分类器进行训练和测试

```
# 使用CountVectorizer将文本转换为词频向量表示
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# 初始化MultinomialNB分类器
classifier = MultinomialNB()

# 在训练集上训练分类器
classifier.fit(X_train_counts, y_train)

# 在测试集上进行预测
y_pred = classifier.predict(X_test_counts)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)

# 计算混淆矩阵
confusion_mat = confusion_matrix(y_test, y_pred)

# 输出分类报告
classification_rep = classification_report(y_test, y_pred)

# 打印结果
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_mat)
print("Classification Report:\n", classification_rep)
```

```
confusion matrix: [[ 562   64   30   29   34  243   15   22   55  157]
 [  36 2201   59   16  120  379   73  150  110  140]
 [  17   49  877    6   34   21    2   42  112   85]
 [  44   45   53  171   88   98   24   13   49  728]
 [  18   87   51   64 1228   25   21   34   43  153]
 [ 228  252   22   59   75 5463  113   31   91  113]
 [  18  128    7    6   55  100  846   18   18   61]
 [  22  123   54   13   60   24   20 1465   58   82]
 [  58   94   93   12   54   73   16   52 1444  119]
 [  94  103  150  312  178   90   41   36  120 2461]]
classification report:                   precision    recall  f1-score   support

      BUSINESS       0.51      0.46      0.49      1211
 ENTERTAINMENT       0.70      0.67      0.68      3284
  FOOD & DRINK       0.63      0.70      0.66      1245
HEALTHY LIVING       0.25      0.13      0.17      1313
     PARENTING       0.64      0.71      0.67      1724
      POLITICS       0.84      0.85      0.84      6447
  QUEER VOICES       0.72      0.67      0.70      1257
STYLE & BEAUTY       0.79      0.76      0.77      1921
        TRAVEL       0.69      0.72      0.70      2015
      WELLNESS       0.60      0.69      0.64      3585

      accuracy                           0.70     24002
     macro avg       0.64      0.64      0.63     24002
  weighted avg       0.69      0.70      0.69     24002

accuracy: 0.6965252895592035
```

5.2 支持向量机（SVM）进行训练和测试

```
# 使用TfidfVectorizer将文本转换为TF-IDF向量表示
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
```

```
X_test_tfidf = vectorizer.transform(X_test)

# 初始化LinearSVC分类器
classifier = LinearSVC()

# 在训练集上训练分类器
classifier.fit(X_train_tfidf, y_train)

# 在测试集上进行预测
y_pred = classifier.predict(X_test_tfidf)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)

# 计算混淆矩阵
confusion_mat = confusion_matrix(y_test, y_pred)

# 输出分类报告
classification_rep = classification_report(y_test, y_pred)

# 打印结果
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_mat)
print("Classification Report:\n", classification_rep)
```

```
confusion matrix: [[ 292   63   10    0   15  497    3   18   46  267]
 [   7 2230   18    0   63  594   20   94   80  178]
 [   3   76  753    0   23   57    0   35  121  177]
 [   6   59   23   13   43  192    6   10   33  928]
 [   3  138   27    4 1027   97    5   31   45  347]
 [  33  168   12    5   32 5948   26   15   53  155]
 [   1  173    4    0   44  263  647   17   12   96]
 [   5  191   23    0   32   54    4 1410   51  151]
 [  10  130   45    0   31  196    5   45 1340  213]
 [  11  104   76    7   82  218    6   24   76 2981]]
classification report:                precision    recall  f1-score   support

      BUSINESS       0.79      0.24      0.37      1211
 ENTERTAINMENT       0.67      0.68      0.67      3284
  FOOD & DRINK       0.76      0.60      0.67      1245
HEALTHY LIVING       0.45      0.01      0.02      1313
     PARENTING       0.74      0.60      0.66      1724
      POLITICS       0.73      0.92      0.82      6447
  QUEER VOICES       0.90      0.51      0.65      1257
STYLE & BEAUTY       0.83      0.73      0.78      1921
        TRAVEL       0.72      0.67      0.69      2015
      WELLNESS       0.54      0.83      0.66      3585

      accuracy                           0.69     24002
     macro avg       0.71      0.58      0.60     24002
  weighted avg       0.70      0.69      0.67     24002

accuracy: 0.6933172235647029
```

5.3 逻辑回归进行训练和测试

```
# 使用TfidfVectorizer将文本转换为TF-IDF向量表示
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

```python
# 初始化LogisticRegression分类器
classifier = LogisticRegression()

# 在训练集上训练分类器
classifier.fit(X_train_tfidf, y_train)

# 在测试集上进行预测
y_pred = classifier.predict(X_test_tfidf)

# 计算准确率
accuracy = accuracy_score(y_test, y_pred)

# 计算混淆矩阵
confusion_mat = confusion_matrix(y_test, y_pred)

# 输出分类报告
classification_rep = classification_report(y_test, y_pred)

# 打印结果
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_mat)
print("Classification Report:\n", classification_rep)
```

```
confusion matrix: [[ 488   87   31   19   28  312    9   20   45  172]
 [  21 2447   42   17   97  364   25   77   88  106]
 [  16   91  847    5   27   39    2   33   91   94]
 [  31  113   44  141   63  126   12   17   38  728]
 [  13   75   35    9 1255   56    9   43   36  193]
 [  97  263   19   23   56 5723   62   19   70  115]
 [  15  125    6    6   38  101  874   16   23   53]
 [  16  127   24    2   32   31    5 1554   49   81]
 [  32  124   66   12   40   89   14   72 1428  138]
 [  56  114   91   56  136  138   13   41  105 2835]]
classification report:                precision    recall  f1-score   support

      BUSINESS       0.62      0.40      0.49      1211
 ENTERTAINMENT       0.69      0.75      0.71      3284
  FOOD & DRINK       0.70      0.68      0.69      1245
HEALTHY LIVING       0.49      0.11      0.18      1313
     PARENTING       0.71      0.73      0.72      1724
      POLITICS       0.82      0.89      0.85      6447
  QUEER VOICES       0.85      0.70      0.77      1257
STYLE & BEAUTY       0.82      0.81      0.82      1921
        TRAVEL       0.72      0.71      0.72      2015
      WELLNESS       0.63      0.79      0.70      3585

      accuracy                           0.73     24002
     macro avg       0.71      0.66      0.66     24002
  weighted avg       0.72      0.73      0.72     24002

accuracy: 0.7329389217565203
```

5.4 对文本进行词干化和词形还原及效果验证

有将近百分之 90 的正确率

```
from nltk.stem import PorterStemmer, WordNetLemmatizer
stop_words = set(stopwords.words('english'))
```

```
def stem_text(rawsentence):
    stemmer = PorterStemmer()
    tokens = word_tokenize(rawsentence)
    stemmed_tokens = [stemmer.stem(word) for word in tokens if word not in stop_words]
    return stemmed_tokens

tfidf_converter = TfidfVectorizer(max_features=1500,min_df=5,max_df=0.7, stop_words=None,toke
X_train_tfidf = tfidf_converter.fit_transform(X_train)
X_test_tfidf = tfidf_converter.transform(X_test)

print(tfidf_converter.get_feature_names())

classifier = MultinomialNB().fit(X_train_tfidf, y_train)
y_pred = classifier.predict(X_test_tfidf)
print('confusion matrix:',confusion_matrix(y_test,y_pred))
print('classification report:', classification_report(y_test,y_pred))
print('accuracy:',accuracy_score(y_test, y_pred))
```

	precision	recall	f1-score	support
ARTS	0.89	0.57	0.70	1509
ARTS & CULTURE	0.93	0.99	0.96	1338
BLACK VOICES	0.93	0.90	0.91	4567
BUSINESS	0.23	0.99	0.38	5989
COLLEGE	0.96	0.80	0.87	1142
COMEDY	0.83	0.82	0.83	5383
CRIME	0.95	0.75	0.84	3558
CULTURE & ARTS	1.00	0.95	0.97	1072
DIVORCE	1.00	1.00	1.00	3426
EDUCATION	0.99	0.88	0.93	1011
ENTERTAINMENT	0.87	0.80	0.83	17265
ENVIRONMENT	1.00	0.92	0.96	1442
FIFTY	0.98	0.74	0.85	1401
FOOD & DRINK	0.99	0.98	0.98	6338
GOOD NEWS	0.92	0.72	0.81	1398
GREEN	0.97	0.78	0.86	2615
HEALTHY LIVING	0.95	0.78	0.86	6676
HOME & LIVING	0.97	0.96	0.97	4319
IMPACT	0.99	0.88	0.93	3481
LATINO VOICES	0.99	0.89	0.94	1124
MEDIA	0.96	0.76	0.85	2931
MONEY	1.00	0.97	0.99	1755
PARENTING	1.00	0.98	0.99	8787
PARENTS	0.93	0.88	0.91	3944
POLITICS	0.95	0.86	0.90	35354
QUEER VOICES	0.97	0.86	0.91	6319
RELIGION	0.98	0.72	0.83	2570
SCIENCE	0.99	0.80	0.88	2201
SPORTS	0.94	0.82	0.87	5057
STYLE	0.90	0.64	0.75	2243
STYLE & BEAUTY	1.00	0.98	0.99	9810
TASTE	0.97	0.89	0.93	2085
TECH	0.98	0.97	0.97	2103
THE WORLDPOST	0.99	1.00	0.99	3664
TRAVEL	1.00	0.95	0.97	9895
U.S. NEWS	0.00	0.00	0.00	1368
WEDDINGS	1.00	1.00	1.00	3653
WEIRD NEWS	0.97	0.77	0.86	2771
WELLNESS	1.00	0.99	1.00	17941
WOMEN	0.99	0.85	0.91	3567
WORLD NEWS	1.00	0.66	0.79	3257
WORLDPOST	0.94	0.48	0.64	2579
accuracy			0.87	208908
macro avg	0.92	0.83	0.86	208908
weighted avg	[illegible]	[illegible]	[illegible]	[illegible]

6 最终选定 knn 模型

6.1 模型的正确率

模型的正确率为百分之 88

```
confusion matrix: [[ 863    0    0 ...    0    0    0]
 [   0 1326    0 ...    0    0    0]
 [   0    1 4089 ...    1    0    2]
 ...
 [   1    0    9 ... 3015    0    0]
 [  25    7   36 ...    1 2137   13]
 [   0    0    0 ...    0    0 1242]]
classification report:                 precision    recall  f1-score   support

          ARTS       0.91      0.57      0.70      1509
ARTS & CULTURE       0.94      0.99      0.96      1338
  BLACK VOICES       0.94      0.90      0.92      4567
      BUSINESS       0.23      0.99      0.38      5989
       COLLEGE       0.96      0.80      0.87      1142
        COMEDY       0.85      0.82      0.84      5383
         CRIME       0.96      0.75      0.85      3558
CULTURE & ARTS       1.00      0.95      0.97      1072
       DIVORCE       1.00      1.00      1.00      3426
     EDUCATION       0.99      0.88      0.93      1011
 ENTERTAINMENT       0.89      0.80      0.84     17265
   ENVIRONMENT       1.00      0.92      0.96      1442
         FIFTY       0.98      0.74      0.85      1401
  FOOD & DRINK       0.99      0.98      0.98      6338
     GOOD NEWS       0.93      0.72      0.81      1398
         GREEN       0.98      0.78      0.87      2615
HEALTHY LIVING       0.95      0.78      0.86      6676
 HOME & LIVING       0.97      0.96      0.97      4319
        IMPACT       0.99      0.88      0.93      3481
 LATINO VOICES       0.99      0.89      0.94      1124
         MEDIA       0.97      0.76      0.85      2931
         MONEY       1.00      0.97      0.99      1755
     PARENTING       1.00      0.98      0.99      8787
       PARENTS       0.94      0.88      0.91      3944
      POLITICS       0.96      0.86      0.91     35354
  QUEER VOICES       0.98      0.86      0.92      6319
      RELIGION       0.98      0.72      0.83      2570
       SCIENCE       0.99      0.80      0.88      2201
        SPORTS       0.95      0.82      0.88      5057
         STYLE       0.90      0.64      0.75      2243
```

```
STYLE & BEAUTY       1.00      0.98      0.99      9810
         TASTE       0.97      0.89      0.93      2085
          TECH       0.98      0.97      0.98      2103
 THE WORLDPOST       0.99      1.00      0.99      3664
        TRAVEL       1.00      0.95      0.97      9895
      WEDDINGS       1.00      1.00      1.00      3653
    WEIRD NEWS       0.97      0.77      0.86      2771
      WELLNESS       1.00      0.99      1.00     17941
         WOMEN       0.99      0.85      0.91      3567
    WORLD NEWS       1.00      0.66      0.79      3257
     WORLDPOST       0.95      0.48      0.64      2579

      accuracy                           0.88    207540
     macro avg       0.95      0.85      0.89    207540
  weighted avg       0.94      0.88      0.90    207540

accuracy_score: 0.8771369374578395
```

6.2 模型的调用方法

```
#参数为json文件的路径
def predict_json(pathToJson):
    df = load_test_set(pathToJson)
    df['short_description'] = df.short_description.apply(process_text)
    df['short_description'] = df.short_description.apply(join_word)
    knn, vec = joblib.load('knn_model.joblib')

    feature = vec.transform(df['short_description'])

    prediction = knn.predict(feature)
```

```
    print('confusion matrix:', confusion_matrix(df['category'], prediction))

    print('classification report:', classification_report(df['category'], prediction))

    print('accuracy_score:', accuracy_score(df['category'], prediction))

#参数为新闻的headline, authors, link, des, date
def predict(headline, authors, link, des, date):
    model, vec = joblib.load('knn_model.joblib')
    feature = vec.transform([des])
    prediction = model.predict(feature)
    print(prediction[0])
```