Environment Setup for AI-Powered Jira Ticket Summarizer & Prioritizer

```
!pip install -q wordcloud textblob spacy plotly xgboost seaborn scikit-learn tensorflow transformers
!python -m spacy download en_core_web_sm

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')

plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

print("🚀 AI-Powered Jira Ticket Summarizer & Prioritizer")
print("=" * 60)
print("✅ Environment setup complete!")
```

Show hidden output

Uploading and Processing Jira Datasets

```
from google.colab import files

import io

print("📤 Upload your Jira dataset files")
print("Supported formats: Excel (.xlsx, .xls), CSV (.csv)")
print("-" * 50)

uploaded = files.upload()
datasets = {}
for filename, content in uploaded.items():
    try:
        if filename.endswith('.xlsx') or filename.endswith('.xls'):
            df = pd.read_excel(io.BytesIO(content))
```

```
            df = pd.read_excel(io.BytesIO(content))
        elif filename.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(content))
        else:
            print(f"⚠️ Unsupported file format: {filename}")
            continue

        datasets[filename] = df
        print(f"✅ Loaded {filename}: {df.shape[0]} rows, {df.shape[1]} columns")

    except Exception as e:
        print(f"❌ Error loading {filename}: {str(e)}")

if datasets:
    main_file = list(datasets.keys())[0]
    jira_data = datasets[main_file]
    print(f"\n🎯 Using {main_file} as main dataset")
    print(f"Dataset shape: {jira_data.shape}")
else:
    print("❌ No valid datasets uploaded!")
```

Show hidden output

Jira Dataset Profiling: Structure, Stats & Missing Values

```
print("📊 DATASET OVERVIEW")
print("=" * 60)

print(f"Dataset Shape: {jira_data.shape}")
print(f"Memory Usage: {jira_data.memory_usage(deep=True).sum() / 1024**2:.2f} MB")

print("\n📋 First 5 rows:")
print(jira_data.head())

print("\n🔍 Data Information:")
print(jira_data.info())

print("\n❌ Missing Values:")
missing_data = jira_data.isnull().sum()
missing_percent = (missing_data / len(jira_data)) * 100
missing_df = pd.DataFrame({
    'Missing Count': missing_data,
    'Percentage': missing_percent
```

```
}).sort_values('Missing Count', ascending=False)
print(missing_df[missing_df['Missing Count'] > 0])

print("\n📈 Statistical Summary:")
print(jira_data.describe(include='all'))
```

```
📊 DATASET OVERVIEW
============================================================
Dataset Shape: (11, 2)
Memory Usage: 0.00 MB

📋 First 5 rows:
                                  <b>Argumentos</b>  \
0  Contribuir para uma apreciação mais realista d...
1  Desenvolver uma visão completa de um problema ...
2  Evitar a duplicação do esforço científico, des...
3  Facilitar e promover a comunicação científica/...
4  Fornecer equilíbrio, indicar imaturidade do co...

                                     <b>Autores</b>
0  (<a href="#B2"> ANDERSON; SPROTT; OLSEN, 2013 ...
1                   (<a href="#B35"> WOLF, 2017 </a>)
2  (<a href="#B2"> ANDERSON; SPROTT; OLSEN, 2013 ...
3  (<a href="#B2"> ANDERSON; SPROTT; OLSEN, 2013 ...
4                 (<a href="#B3"> ASHLEY, 2004 </a>)

🔍 Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   <b>Argumentos</b>  11 non-null     object
 1   <b>Autores</b>     11 non-null     object
dtypes: object(2)
memory usage: 308.0+ bytes
None

❌ Missing Values:
Empty DataFrame
Columns: [Missing Count, Percentage]
Index: []

📈 Statistical Summary:
                                       <b>Argumentos</b>  \
count                                                 11
unique                                                11
top     Contribuir para uma apreciação mais realista d...
freq                                                   1
```

```
                                          <b>Autores</b>
count                                                11
unique                                               10
top     (<a href="#B2"> ANDERSON; SPROTT; OLSEN, 2013 ...
freq                                                  2
```

Advanced Text Feature Engineering for Jira Tickets

```
import nltk
import spacy
from textblob import TextBlob
from nltk.corpus import stopwords
from nltk.sentiment import SentimentIntensityAnalyzer
import re

nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('vader_lexicon', quiet=True)

nlp = spacy.load('en_core_web_sm')

class AdvancedJiraProcessor:
    def __init__(self):
        self.stop_words = set(stopwords.words('english'))
        self.sia = SentimentIntensityAnalyzer()

    def clean_text(self, text):
        """Advanced text cleaning"""
        if pd.isna(text) or text == '':
            return ""

        text = str(text).lower()
        text = re.sub(r'http\S+|www\S+|https\S+', '', text)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def extract_advanced_features(self, df):
        """Extract comprehensive features"""
        df = df.copy()
```

```
        text_columns = []
        for col in df.columns:
            if df[col].dtype == 'object':
                if df[col].str.len().mean() > 10:
                    text_columns.append(col)

        print(f"🔤 Identified text columns: {text_columns}")
        if text_columns:
            df['combined_text'] = df[text_columns].apply(
                lambda x: ' '.join(x.dropna().astype(str)), axis=1
            )
        else:
            df['combined_text'] = 'sample text for analysis'

        df['combined_text_clean'] = df['combined_text'].apply(self.clean_text)

        df['text_length'] = df['combined_text'].str.len()
        df['word_count'] = df['combined_text_clean'].apply(lambda x: len(x.split()) if x else 0)
        df['char_count'] = df['combined_text_clean'].str.len()

        sentiments = df['combined_text_clean'].apply(self.get_sentiment)
        df['sentiment_compound'] = [s['compound'] for s in sentiments]
        df['sentiment_positive'] = [s['pos'] for s in sentiments]
        df['sentiment_negative'] = [s['neg'] for s in sentiments]
        df['sentiment_neutral'] = [s['neu'] for s in sentiments]

        df['keywords'] = df['combined_text_clean'].apply(self.extract_keywords)
        df['keyword_count'] = df['keywords'].apply(len)

        priority_cols = [col for col in df.columns if 'priority' in col.lower()]
        if priority_cols:
            df['priority_encoded'] = pd.Categorical(df[priority_cols[0]]).codes
        else:
            df['priority'] = np.random.choice(['High', 'Medium', 'Low', 'Critical'], len(df))
            df['priority_encoded'] = pd.Categorical(df['priority']).codes

        return df

    def get_sentiment(self, text):
        """Get sentiment scores"""
        if not text:
            return {'compound': 0, 'pos': 0, 'neg': 0, 'neu': 1}
        return self.sia.polarity_scores(text)

    def extract_keywords(self, text):
```

```
        """Extract keywords using spaCy"""
        if not text:
            return []

        doc = nlp(text)
        keywords = []

        for token in doc:
            if (token.pos_ in ['NOUN', 'ADJ', 'VERB'] and
                not token.is_stop and
                not token.is_punct and
                len(token.text) > 2):
                keywords.append(token.lemma_)

        return list(set(keywords))

processor = AdvancedJiraProcessor()
jira_processed = processor.extract_advanced_features(jira_data)

print("✅ Advanced preprocessing completed!")
print(f"📊 Processed dataset shape: {jira_processed.shape}")
print(f"🔧 New features added: {jira_processed.shape[1] - jira_data.shape[1]}")
```

```
🔤 Identified text columns: ['<b>Argumentos</b>', '<b>Autores</b>']
✅ Advanced preprocessing completed!
📊 Processed dataset shape: (11, 15)
🔧 New features added: 13
```

Jira Data Snapshot: Structure, Quality & Memory Insights

```
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Dataset Shape', 'Missing Values', 'Data Types', 'Memory Usage'),
    specs=[[{"type": "indicator"}, {"type": "bar"}],
           [{"type": "pie"}, {"type": "indicator"}]]
)

fig.add_trace(go.Indicator(
    mode="number",
    value=jira_processed.shape[0],
    title={"text": "Total Records"},
    number={'font': {'size': 40}}
), row=1, col=1)
```

```
missing_data = jira_processed.isnull().sum()
missing_data = missing_data[missing_data > 0]
if not missing_data.empty:
    fig.add_trace(go.Bar(
        x=missing_data.index,
        y=missing_data.values,
        name="Missing Values",
        marker_color='red'
    ), row=1, col=2)

dtype_counts = jira_processed.dtypes.value_counts()
fig.add_trace(go.Pie(
    labels=dtype_counts.index.astype(str),
    values=dtype_counts.values,
    name="Data Types"
), row=2, col=1)

memory_mb = jira_processed.memory_usage(deep=True).sum() / 1024**2
fig.add_trace(go.Indicator(
    mode="number+gauge",
    value=memory_mb,
    title={"text": "Memory (MB)"},
    gauge={'axis': {'range': [0, memory_mb*2]}}
), row=2, col=2)

fig.update_layout(height=600, title_text="📊 Dataset Overview Dashboard")
fig.show()
```

Dataset Overview Dashboard

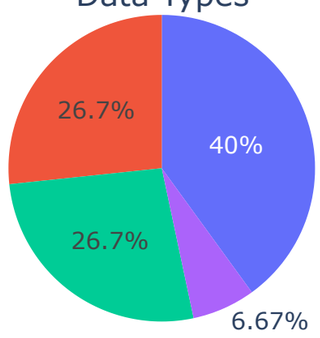
Dataset Shape

Total Records

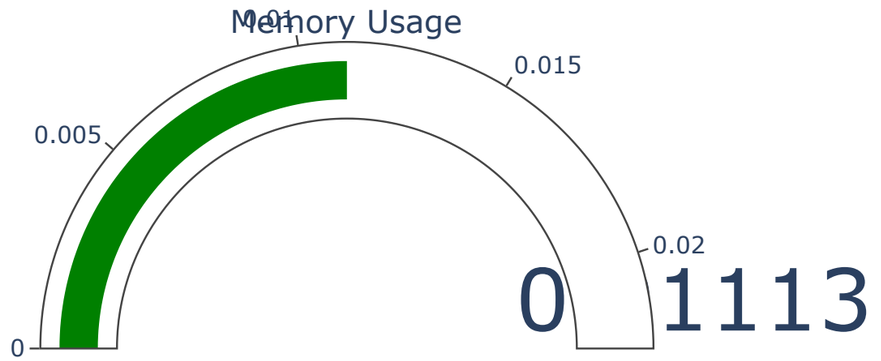
11

Missing Values

Data Types



Memory (MB)

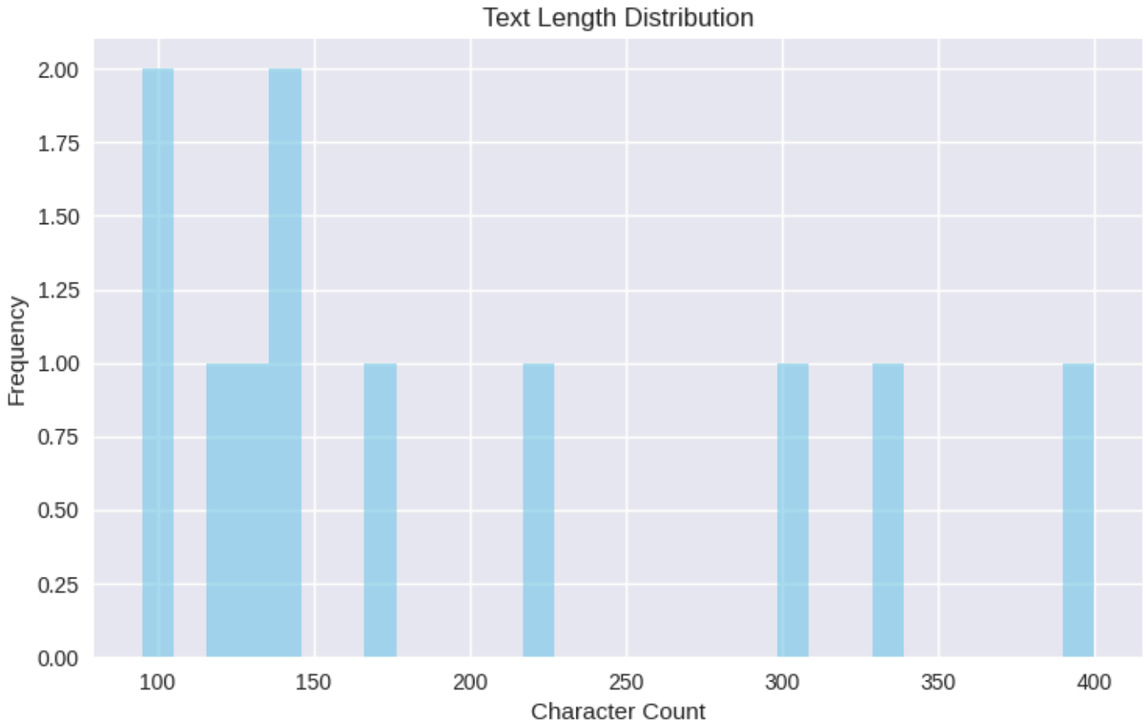


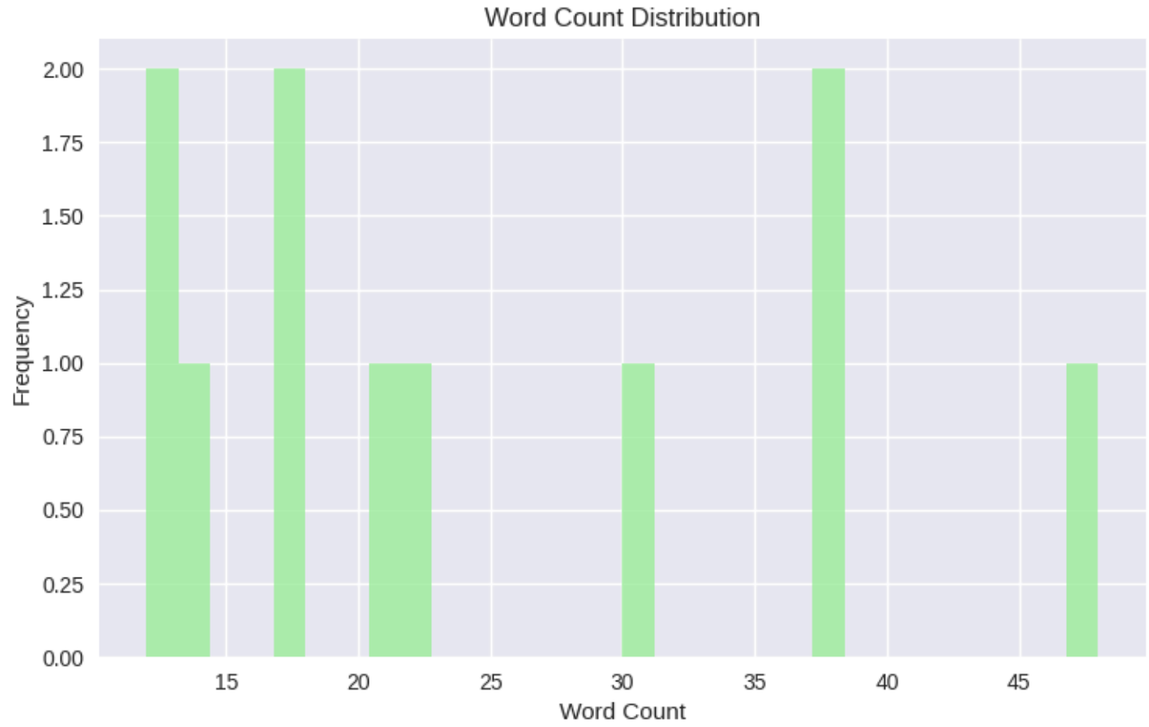
Distribution Insights from Jira Text Features

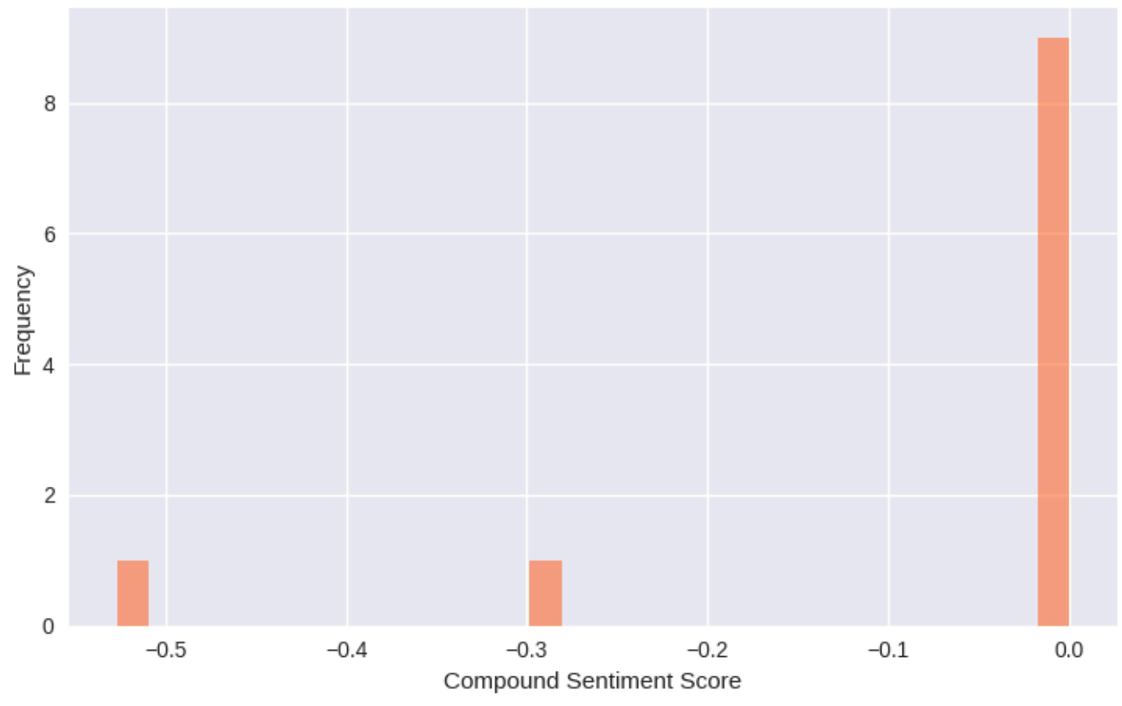
```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('📝 Text Analysis Dashboard', fontsize=16, fontweight='bold')

axes[0,0].hist(jira_processed['text_length'], bins=30, alpha=0.7, color='skyblue')
axes[0,0].set_title('Text Length Distribution')
axes[0,0].set_xlabel('Character Count')
axes[0,0].set_ylabel('Frequency')
```

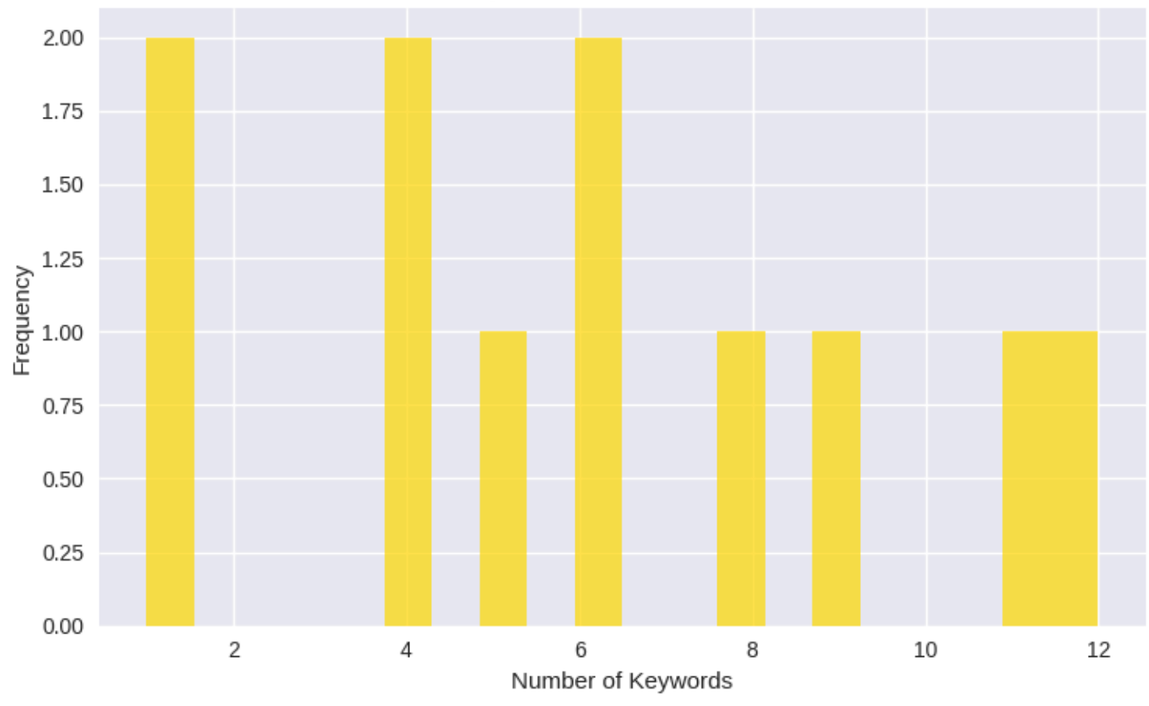
```
axes[0,1].hist(jira_processed['word_count'], bins=30, alpha=0.7, color='lightgreen')
axes[0,1].set_title('Word Count Distribution')
axes[0,1].set_xlabel('Word Count')
axes[0,1].set_ylabel('Frequency')

axes[1,0].hist(jira_processed['sentiment_compound'], bins=30, alpha=0.7, color='coral')
axes[1,0].set_title('Sentiment Score Distribution')
axes[1,0].set_xlabel('Compound Sentiment Score')
axes[1,0].set_ylabel('Frequency')

axes[1,1].hist(jira_processed['keyword_count'], bins=20, alpha=0.7, color='gold')
axes[1,1].set_title('Keywords Count Distribution')
axes[1,1].set_xlabel('Number of Keywords')
axes[1,1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```

Text Analysis Dashboard





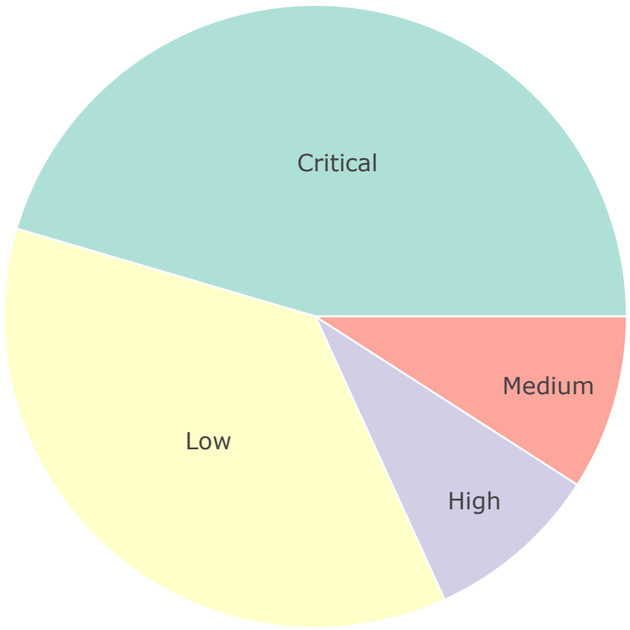




Priority Distribution Analysis

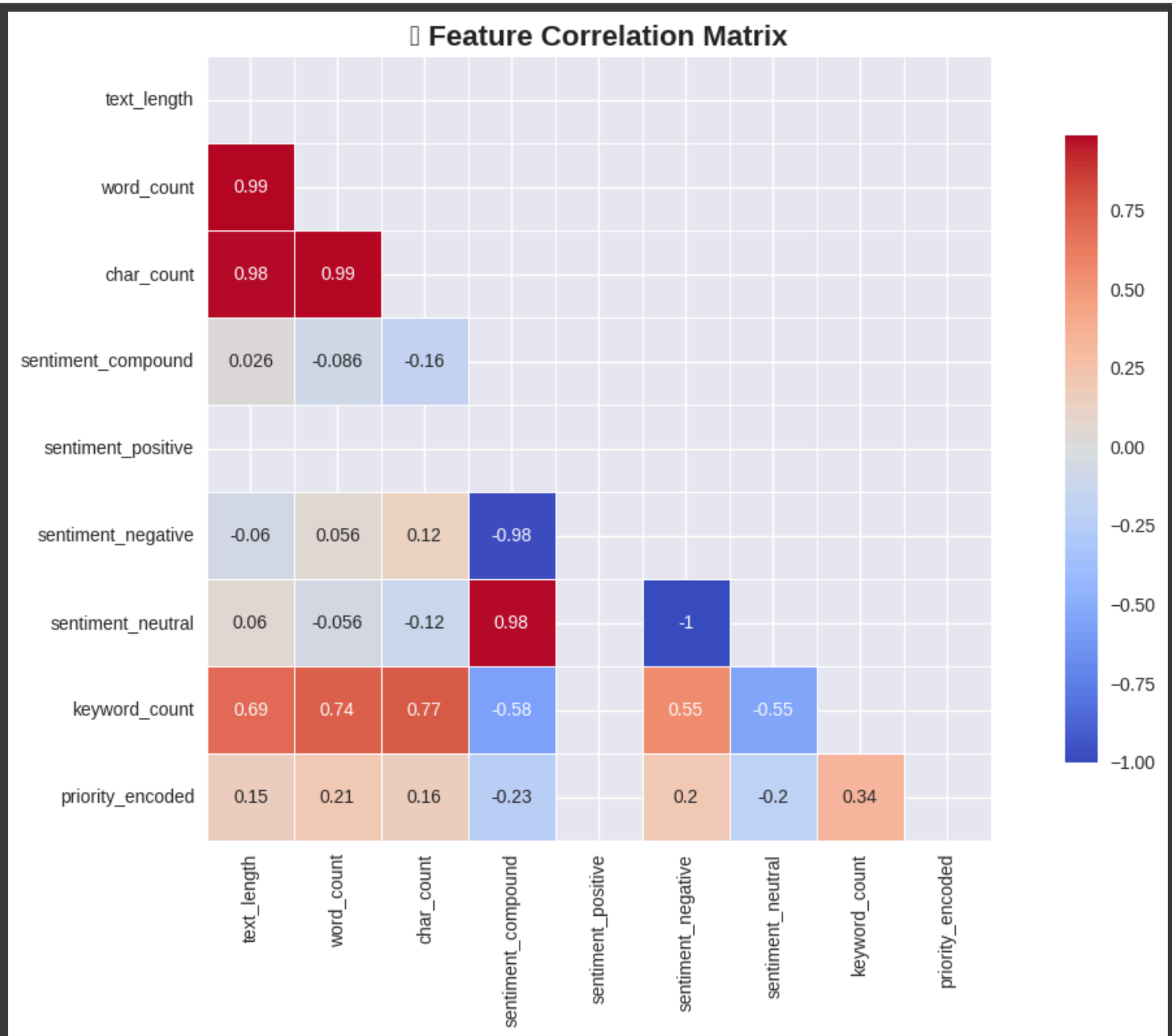
```
priority_col = [col for col in jira_processed.columns if 'priority' in col.lower() and col != 'priority_encoded'][0] if [col for col in jira_processed.columns

fig = px.sunburst(
    jira_processed,
    path=[priority_col],
    title="🎯 Priority Distribution",
    color_discrete_sequence=px.colors.qualitative.Set3
)
fig.update_layout(height=500)
fig.show()
```

🎯 Priority Distribution



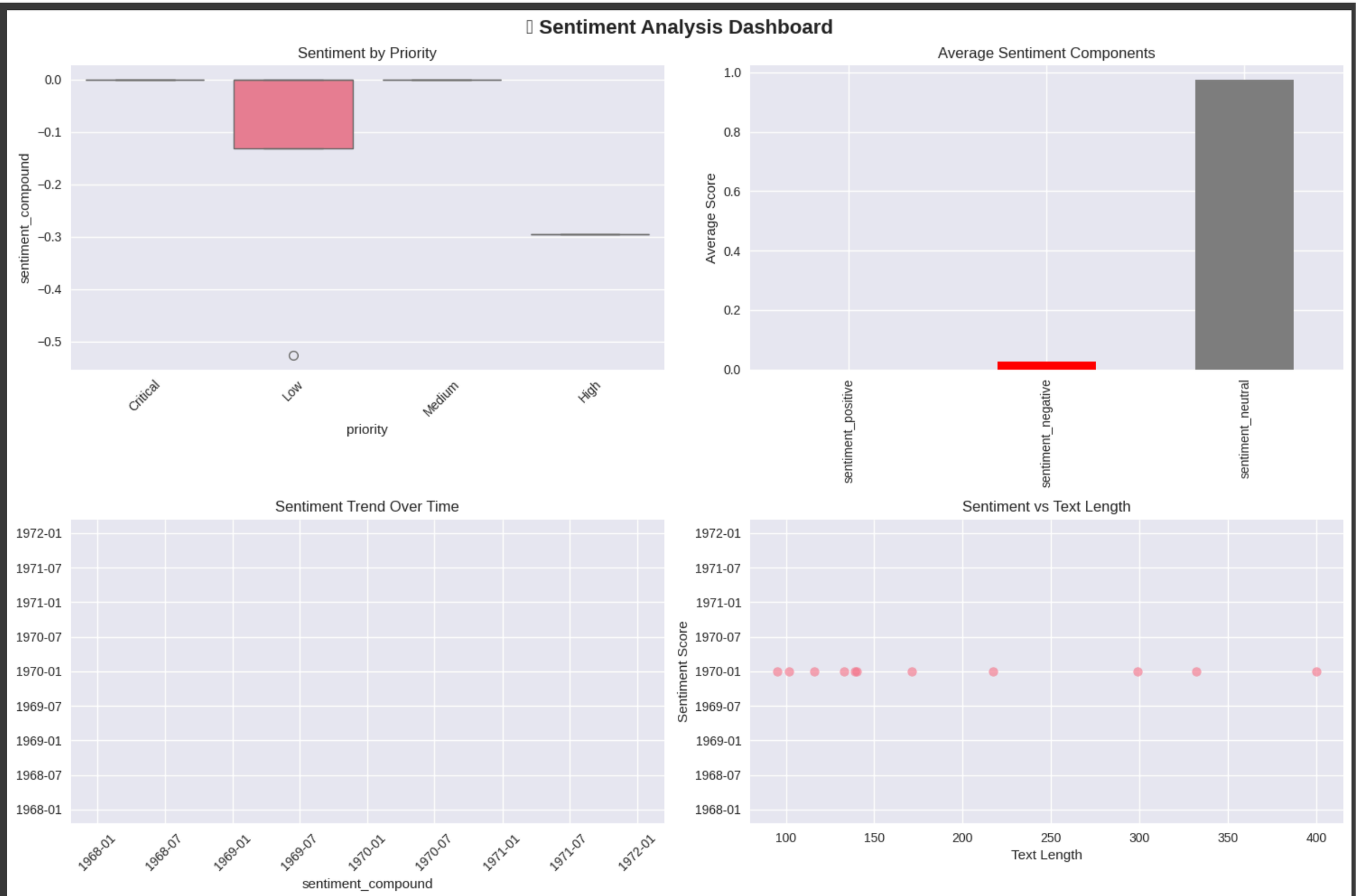
Correlation Heatmap

```
numeric_cols = jira_processed.select_dtypes(include=[np.number]).columns
correlation_matrix = jira_processed[numeric_cols].corr()

plt.figure(figsize=(12, 8))
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=0.5, cbar_kws={"shrink": .8})
plt.title('🔗 Feature Correlation Matrix', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```


Feature Correlation Matrix
text_length
word_count
char_count
sentiment_compound
sentiment_positive
sentiment_negative
sentiment_neutral
keyword_count
priority_encoded
0.99
0.98 0.99
0.026 -0.086 -0.16
-0.06 0.056 0.12 -0.98
0.06 -0.056 -0.12 0.98 -1
0.69 0.74 0.77 -0.58 0.55 -0.55
0.15 0.21 0.16 -0.23 0.2 -0.2 0.34
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Advanced Sentiment Analysis

```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('😊 Sentiment Analysis Dashboard', fontsize=16, fontweight='bold')

sns.boxplot(data=jira_processed, x=priority_col, y='sentiment_compound', ax=axes[0,0])
axes[0,0].set_title('Sentiment by Priority')
axes[0,0].tick_params(axis='x', rotation=45)

sentiment_cols = ['sentiment_positive', 'sentiment_negative', 'sentiment_neutral']
jira_processed[sentiment_cols].mean().plot(kind='bar', ax=axes[0,1], color=['green', 'red', 'gray'])
axes[0,1].set_title('Average Sentiment Components')
axes[0,1].set_ylabel('Average Score')

date_cols = [col for col in jira_processed.columns if 'date' in col.lower() or 'time' in col.lower()]
if date_cols:
    jira_processed[date_cols[0]] = pd.to_datetime(jira_processed[date_cols[0]], errors='coerce')
    daily_sentiment = jira_processed.groupby(jira_processed[date_cols[0]].dt.date)['sentiment_compound'].mean()
    daily_sentiment.plot(ax=axes[1,0], color='blue', alpha=0.7)
    axes[1,0].set_title('Sentiment Trend Over Time')
    axes[1,0].tick_params(axis='x', rotation=45)
else:
    axes[1,0].text(0.5, 0.5, 'No Date Column Found', ha='center', va='center', transform=axes[1,0].transAxes)
    axes[1,0].set_title('Sentiment Trend Over Time')

axes[1,1].scatter(jira_processed['text_length'], jira_processed['sentiment_compound'], alpha=0.6)
axes[1,1].set_xlabel('Text Length')
axes[1,1].set_ylabel('Sentiment Score')
axes[1,1].set_title('Sentiment vs Text Length')

plt.tight_layout()
plt.show()

print("✅ First 5 visualizations completed!")
```



✅ First 5 visualizations completed!

Word Cloud Analysis

```
from wordcloud import WordCloud

print("☁️ Generating Word Clouds...")

fig, axes = plt.subplots(2, 2, figsize=(20, 15))
fig.suptitle('☁️ Word Cloud Analysis', fontsize=20, fontweight='bold')

all_text = ' '.join(jira_processed['combined_text_clean'].dropna())
if all_text.strip():
    wordcloud_all = WordCloud(width=400, height=300, background_color='white').generate(all_text)
    axes[0,0].imshow(wordcloud_all, interpolation='bilinear')
    axes[0,0].set_title('Overall Word Cloud', fontsize=14)
    axes[0,0].axis('off')

priorities = jira_processed[priority_col].unique()
for i, priority in enumerate(priorities[:3]):
    if i < 3:
        row, col = divmod(i+1, 2)
        if row < 2:
            priority_text = ' '.join(
                jira_processed[jira_processed[priority_col] == priority]['combined_text_clean'].dropna()
            )
            if priority_text.strip():
                wordcloud_priority = WordCloud(width=400, height=300, background_color='white').generate(priority_text)
                axes[row, col].imshow(wordcloud_priority, interpolation='bilinear')
                axes[row, col].set_title(f'{priority} Priority Word Cloud', fontsize=14)
                axes[row, col].axis('off')

plt.tight_layout()
plt.show()
```

Generating Word Clouds...

📊 Word Cloud Analysis

Overall Word Cloud

Critical Priority Word Cloud

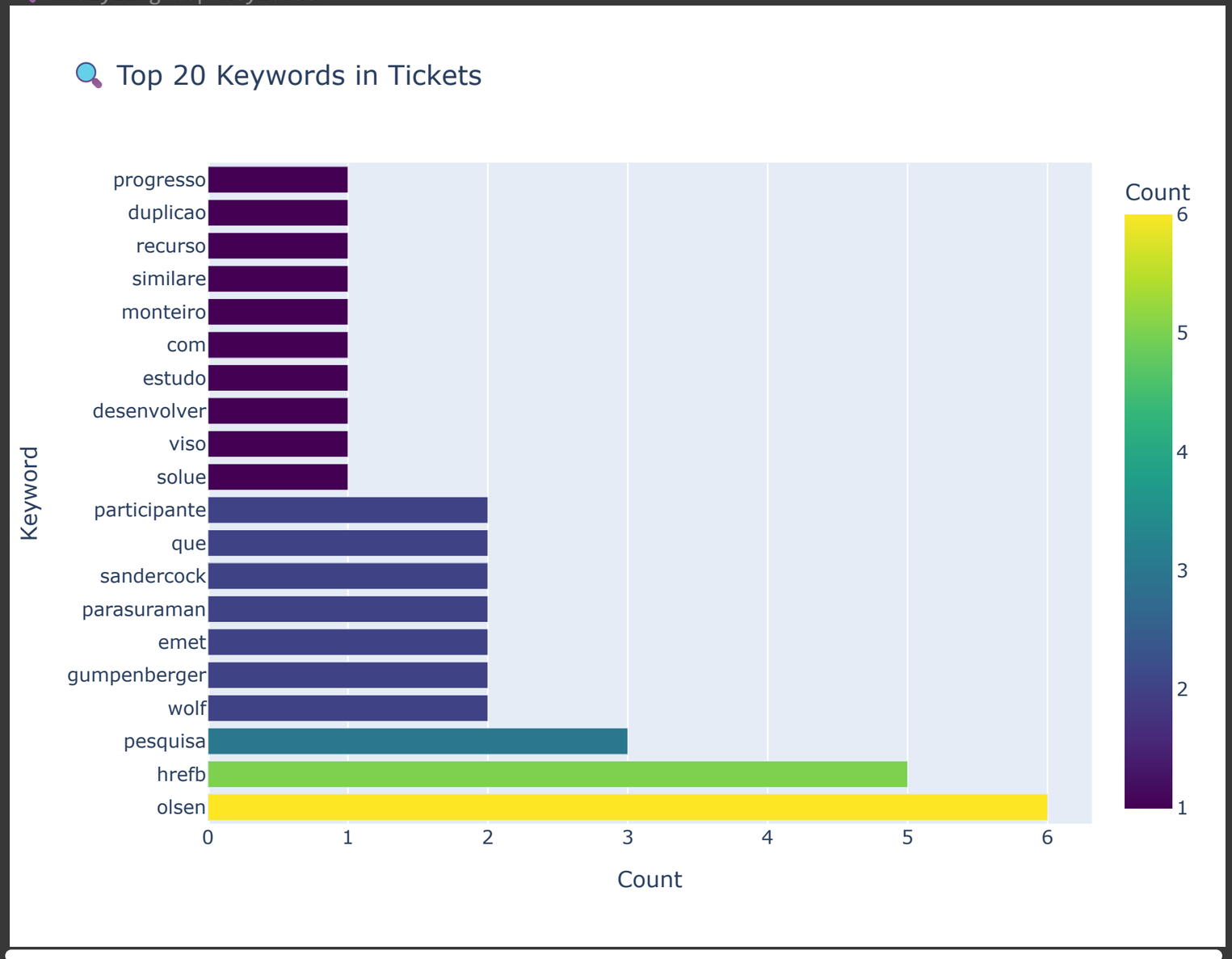
Low Priority Word Cloud

Medium Priority Word Cloud

Top Keywords Analysis

```
print("🔍 Analyzing top keywords...")

from collections import Counter

all_keywords = []
for keywords in jira_processed['keywords']:
    all_keywords.extend(keywords)

keyword_counts = Counter(all_keywords)
top_keywords = keyword_counts.most_common(20)

if top_keywords:
    keywords_df = pd.DataFrame(top_keywords, columns=['Keyword', 'Count'])

    fig = px.bar(
        keywords_df,
        x='Count',
        y='Keyword',
        orientation='h',
        title='🔍 Top 20 Keywords in Tickets',
        color='Count',
        color_continuous_scale='viridis'
    )
    fig.update_layout(height=600)
    fig.show()
```

```
🔍 Analyzing top keywords...
```

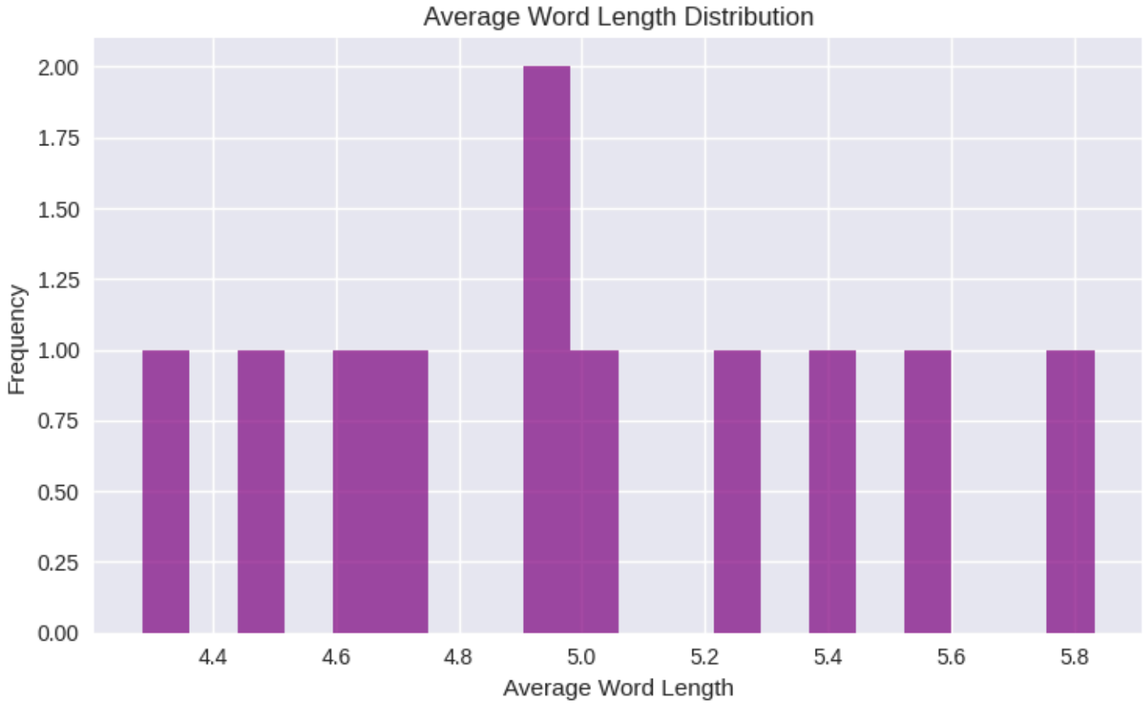


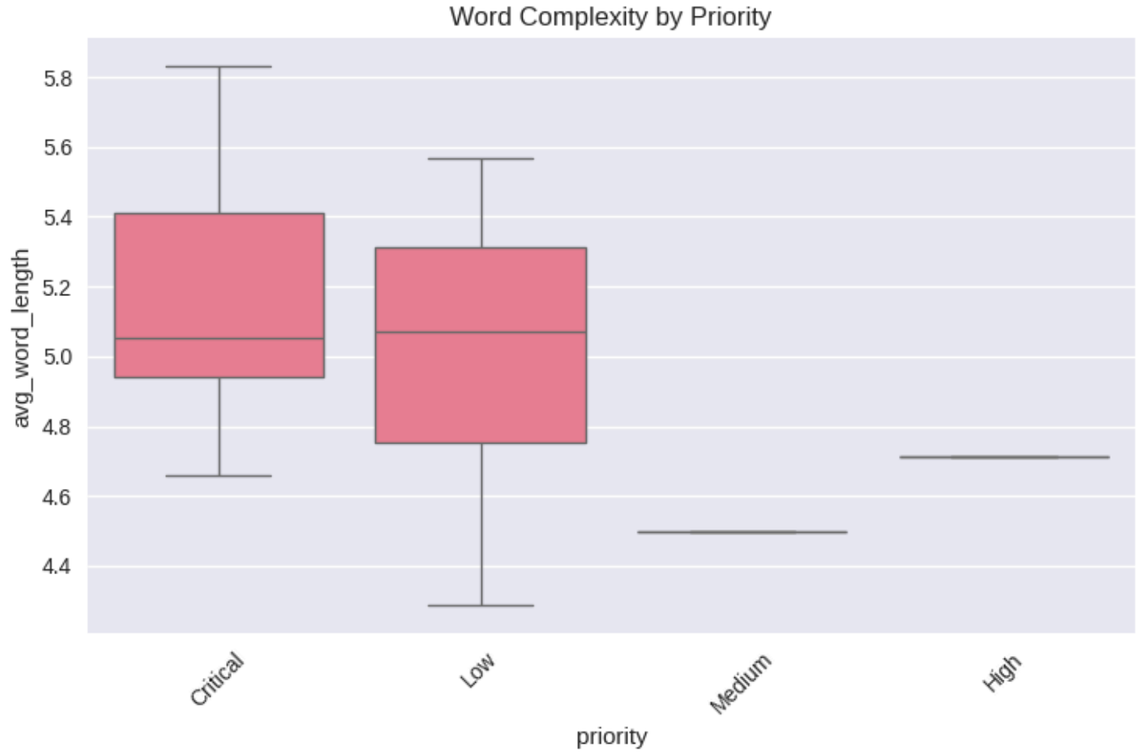
Text Complexity Analysis

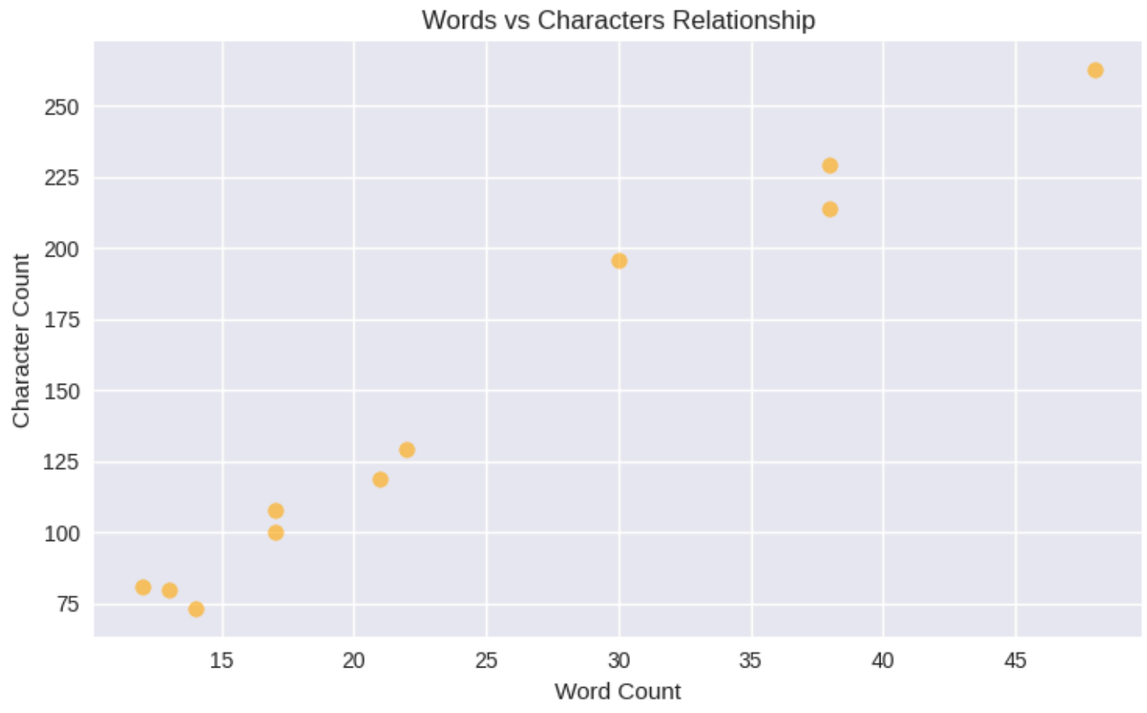
```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('📊 Text Complexity Analysis', fontsize=16, fontweight='bold')

jira_processed['avg_word_length'] = jira_processed['combined_text_clean'].apply(
    lambda x: np.mean([len(word) for word in x.split()]) if x and x.split() else 0
)
```

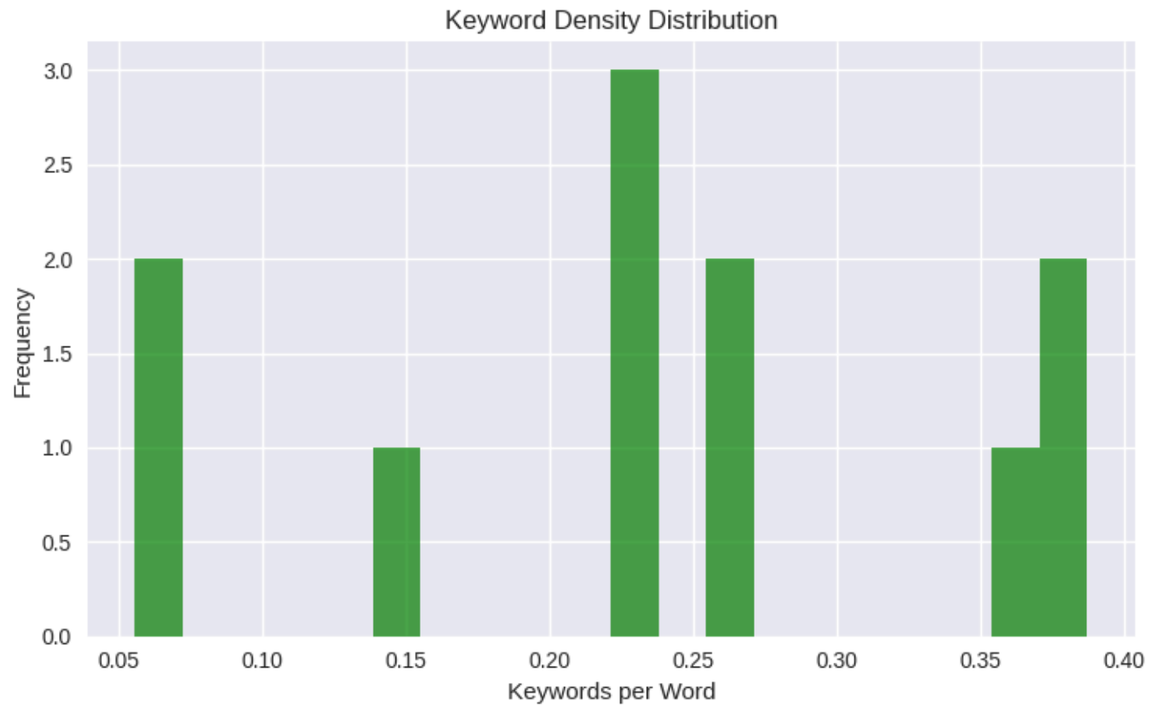
```
axes[0,0].hist(jira_processed['avg_word_length'], bins=20, alpha=0.7, color='purple')
axes[0,0].set_title('Average Word Length Distribution')
axes[0,0].set_xlabel('Average Word Length')
axes[0,0].set_ylabel('Frequency')

sns.boxplot(data=jira_processed, x=priority_col, y='avg_word_length', ax=axes[0,1])
axes[0,1].set_title('Word Complexity by Priority')
axes[0,1].tick_params(axis='x', rotation=45)

axes[1,0].scatter(jira_processed['word_count'], jira_processed['char_count'], alpha=0.6, color='orange')
axes[1,0].set_xlabel('Word Count')
axes[1,0].set_ylabel('Character Count')
axes[1,0].set_title('Words vs Characters Relationship')

jira_processed['keyword_density'] = jira_processed['keyword_count'] / (jira_processed['word_count'] + 1)
axes[1,1].hist(jira_processed['keyword_density'], bins=20, alpha=0.7, color='green')
axes[1,1].set_title('Keyword Density Distribution')
axes[1,1].set_xlabel('Keywords per Word')
axes[1,1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```

Text Complexity Analysis







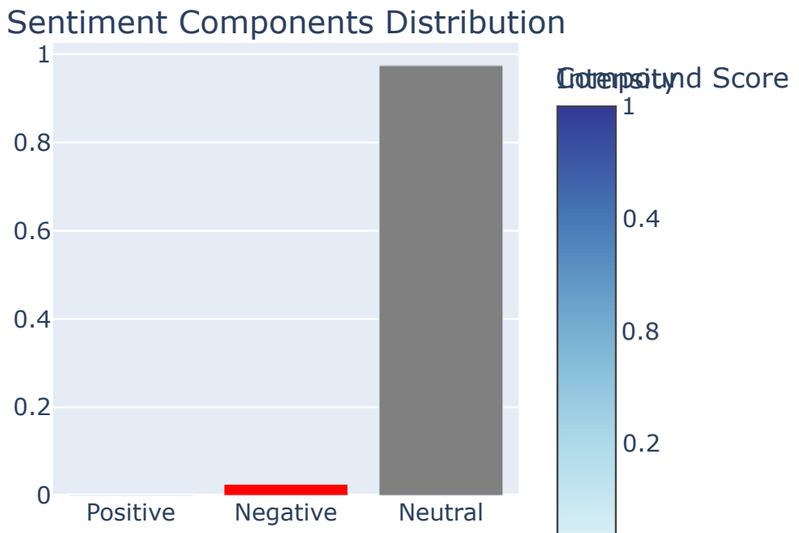


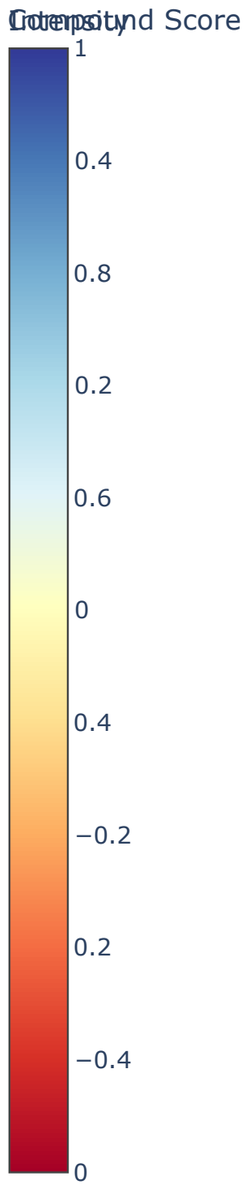
Sentiment Deep Dive

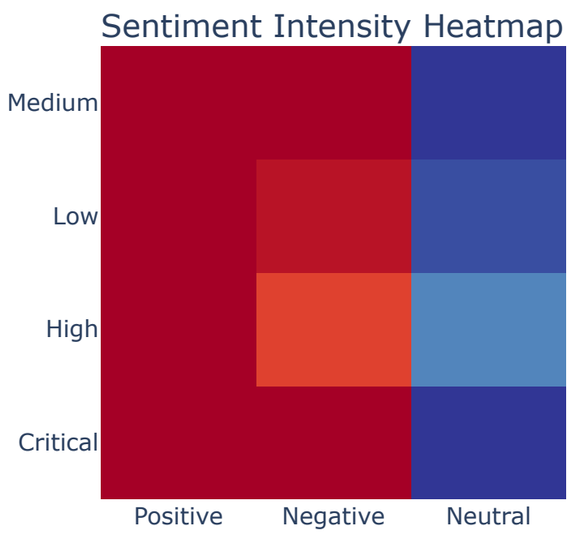
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import pandas as pd

jira_processed['sentiment_compound'] = pd.to_numeric(jira_processed['sentiment_compound'], errors='coerce')

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=[
        'Sentiment by Priority',
        'Sentiment Components Distribution',
        'Sentiment Intensity Heatmap',
        'Positive vs Negative Sentiment'
    ],
    specs=[[{"type": "violin"}, {"type": "bar"}],
           [{"type": "heatmap"}, {"type": "scatter"}]]
)

for priority in jira_processed[priority_col].dropna().unique():
    sentiment_data = jira_processed[jira_processed[priority_col] == priority]['sentiment_compound']
    fig.add_trace(go.Violin(
        y=sentiment_data,
        name=str(priority),
        box_visible=True,
        meanline_visible=True,
        line_color='blue'
    ), row=1, col=1)

sentiment_means = jira_processed[['sentiment_positive', 'sentiment_negative', 'sentiment_neutral']].mean()
fig.add_trace(go.Bar(
    x=['Positive', 'Negative', 'Neutral'],
    y=sentiment_means.values,
    marker_color=['green', 'red', 'gray'],
    name='Average Sentiment'
), row=1, col=2)
```

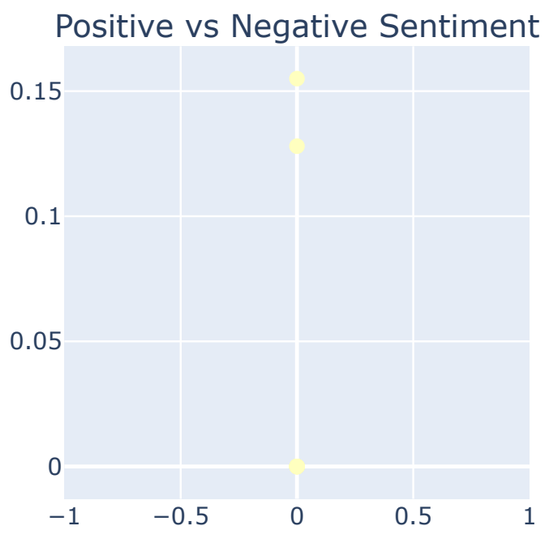
```
sentiment_by_priority = jira_processed.groupby(priority_col)[
    ['sentiment_positive', 'sentiment_negative', 'sentiment_neutral']
].mean().dropna()

fig.add_trace(go.Heatmap(
    z=sentiment_by_priority.values,
    x=['Positive', 'Negative', 'Neutral'],
    y=sentiment_by_priority.index.astype(str),
    colorscale='RdYlBu',
    colorbar=dict(title='Intensity')
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=jira_processed['sentiment_positive'],
    y=jira_processed['sentiment_negative'],
    mode='markers',
    marker=dict(
        size=8,
        color=jira_processed['sentiment_compound'],
        colorscale='RdYlBu',
        showscale=True,
        colorbar=dict(title='Compound Score')
    ),
    text=jira_processed[priority_col].astype(str),
    hovertemplate='Priority: %{text}<br>Positive: %{x}<br>Negative: %{y}<extra></extra>'
), row=2, col=2)

fig.update_layout(
    height=800,
    title_text="😊 Advanced Sentiment Analysis Dashboard",
    showlegend=False
)
fig.show()
```

😊 Advanced Sentiment Analysis Dashboard





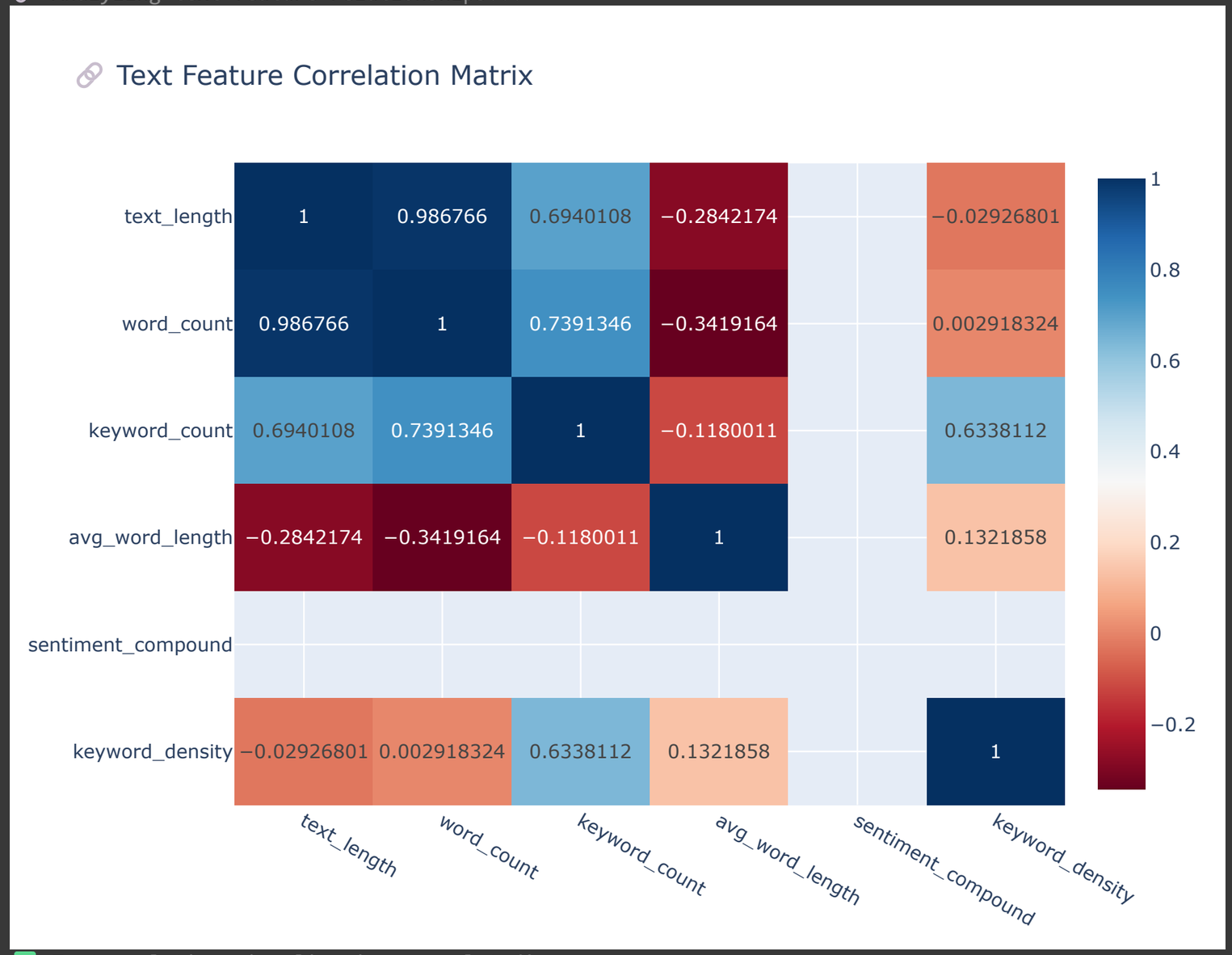




Text Feature Relationships

```
print("🔗 Analyzing text feature relationships...")
text_features = ['text_length', 'word_count', 'keyword_count', 'avg_word_length',
                 'sentiment_compound', 'keyword_density']
feature_matrix = jira_processed[text_features].corr()

fig = px.imshow(feature_matrix,
                text_auto=True,
                aspect="auto",
                color_continuous_scale='RdBu',
                title="🔗 Text Feature Correlation Matrix")
fig.update_layout(height=600)
fig.show()

print("✅ Text analytics visualizations completed!")
```

```
Analyzing text feature relationships...
```




```
Text analytics visualizations completed!
```

Priority Prediction Pipeline for Jira Tickets

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
import xgboost as xgb
import pandas as pd
import numpy as np
from scipy.sparse import hstack

class JiraMLPipeline:
    def __init__(self):
        self.models = {}
        self.vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
        self.scaler = StandardScaler()
        self.feature_names = []

    def prepare_features(self, df):
        """Prepare features for ML models"""
        text_features = self.vectorizer.fit_transform(df['combined_text_clean'])

        numerical_features = [
            'text_length', 'word_count', 'keyword_count',
            'avg_word_length', 'sentiment_compound',
            'sentiment_positive', 'sentiment_negative'
        ]

        for feature in numerical_features:
            if feature not in df.columns:
                df[feature] = 0

        numerical_data = self.scaler.fit_transform(df[numerical_features])
        combined_features = hstack([text_features, numerical_data]).tocsr()
        self.feature_names = list(self.vectorizer.get_feature_names_out()) + numerical_features
        return combined_features

    def train_models(self, X, y):
        """Train multiple ML models"""
        y_series = pd.Series(y)
        class_counts = y_series.value_counts()
        valid_classes = class_counts[class_counts >= 2].index
        valid_indices = y_series.isin(valid_classes).values

        X = X[valid_indices]
        y = y[valid_indices]
        y = LabelEncoder().fit_transform(y)

        test_size = max(0.4, len(set(y)) / len(y))
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42, stratify=y
```

```
        )

        class_counts_train = pd.Series(y_train).value_counts()
        min_class_size = class_counts_train.min()
        n_splits = min(5, min_class_size)
        skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

        models = {
            'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
            'XGBoost': xgb.XGBClassifier(random_state=42, eval_metric='mlogloss'),
            'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
            'Gradient Boosting': GradientBoostingClassifier(random_state=42)
        }

        results = {}

        for name, model in models.items():
            print(f"🔄 Training {name}...")
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            cv_scores = cross_val_score(model, X_train, y_train, cv=skf)

            results[name] = {
                'model': model,
                'accuracy': accuracy,
                'cv_mean': cv_scores.mean(),
                'cv_std': cv_scores.std(),
                'predictions': y_pred,
                'test_labels': y_test,
                'X_test': X_test
            }

            print(f"✅ {name}: Accuracy = {accuracy:.3f}, CV = {cv_scores.mean():.3f} ± {cv_scores.std():.3f}")

        self.models = results
        return results

print("🔄 Preparing data for machine learning...")

ml_pipeline = JiraMLPipeline()
X = ml_pipeline.prepare_features(jira_processed)
y = jira_processed['priority_encoded'].values
print(f"📊 Feature matrix shape: {X.shape}")
print(f"🎯 Target classes: {sorted(jira_processed['priority_encoded'].unique())}")
```

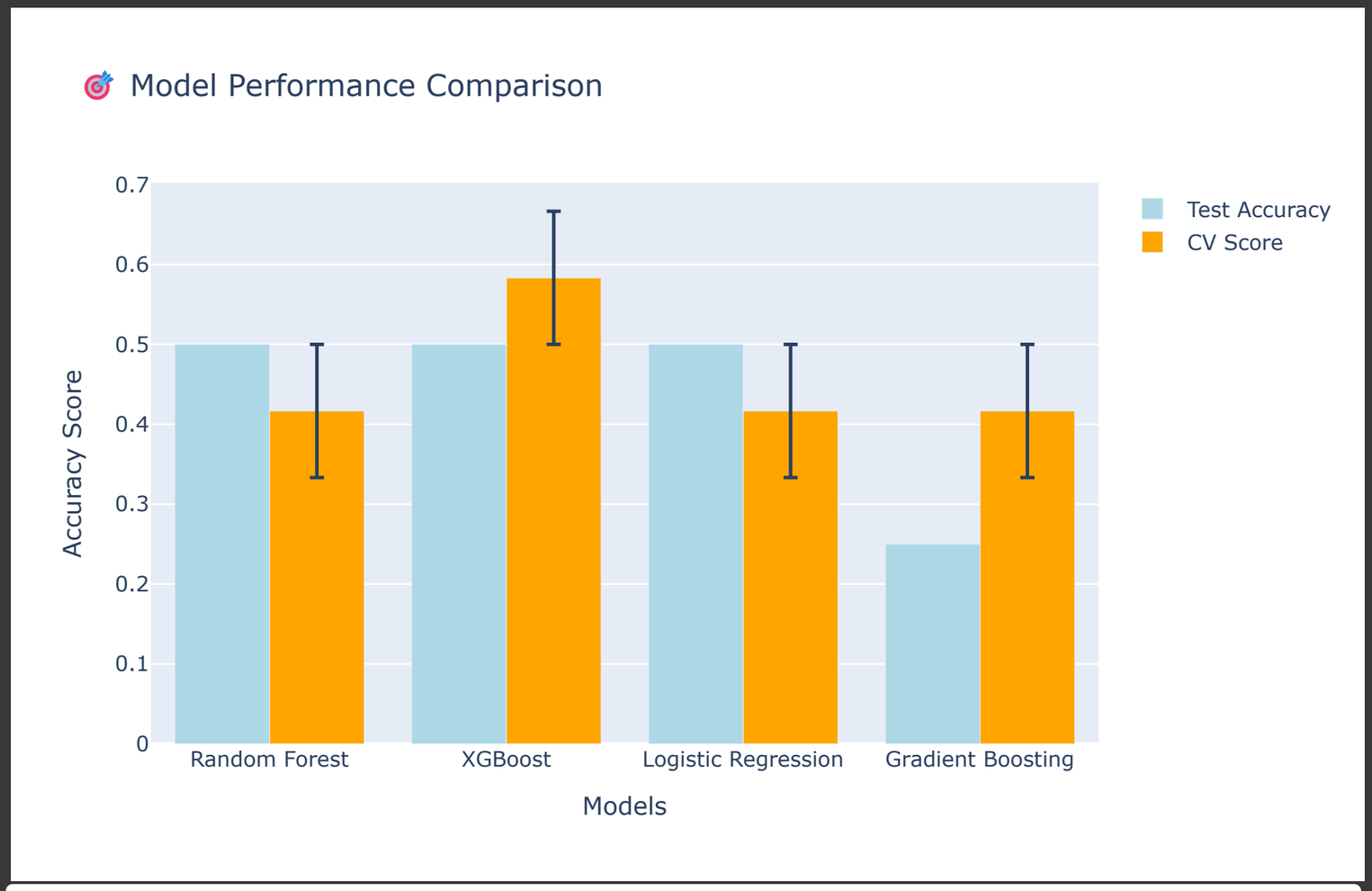
```
print("\n🤖 Training machine learning models...")
model_results = ml_pipeline.train_models(X, y)
print("\n✅ Machine learning pipeline completed!")
```

```
🔄 Preparing data for machine learning...
📊 Feature matrix shape: (11, 120)
🎯 Target classes: [np.int8(0), np.int8(1), np.int8(2), np.int8(3)]

🤖 Training machine learning models...
🔄 Training Random Forest...
✅ Random Forest: Accuracy = 0.500, CV = 0.417 ± 0.083
🔄 Training XGBoost...
✅ XGBoost: Accuracy = 0.500, CV = 0.583 ± 0.083
🔄 Training Logistic Regression...
✅ Logistic Regression: Accuracy = 0.500, CV = 0.417 ± 0.083
🔄 Training Gradient Boosting...
✅ Gradient Boosting: Accuracy = 0.250, CV = 0.417 ± 0.083

✅ Machine learning pipeline completed!
```

Model Performance Comparison

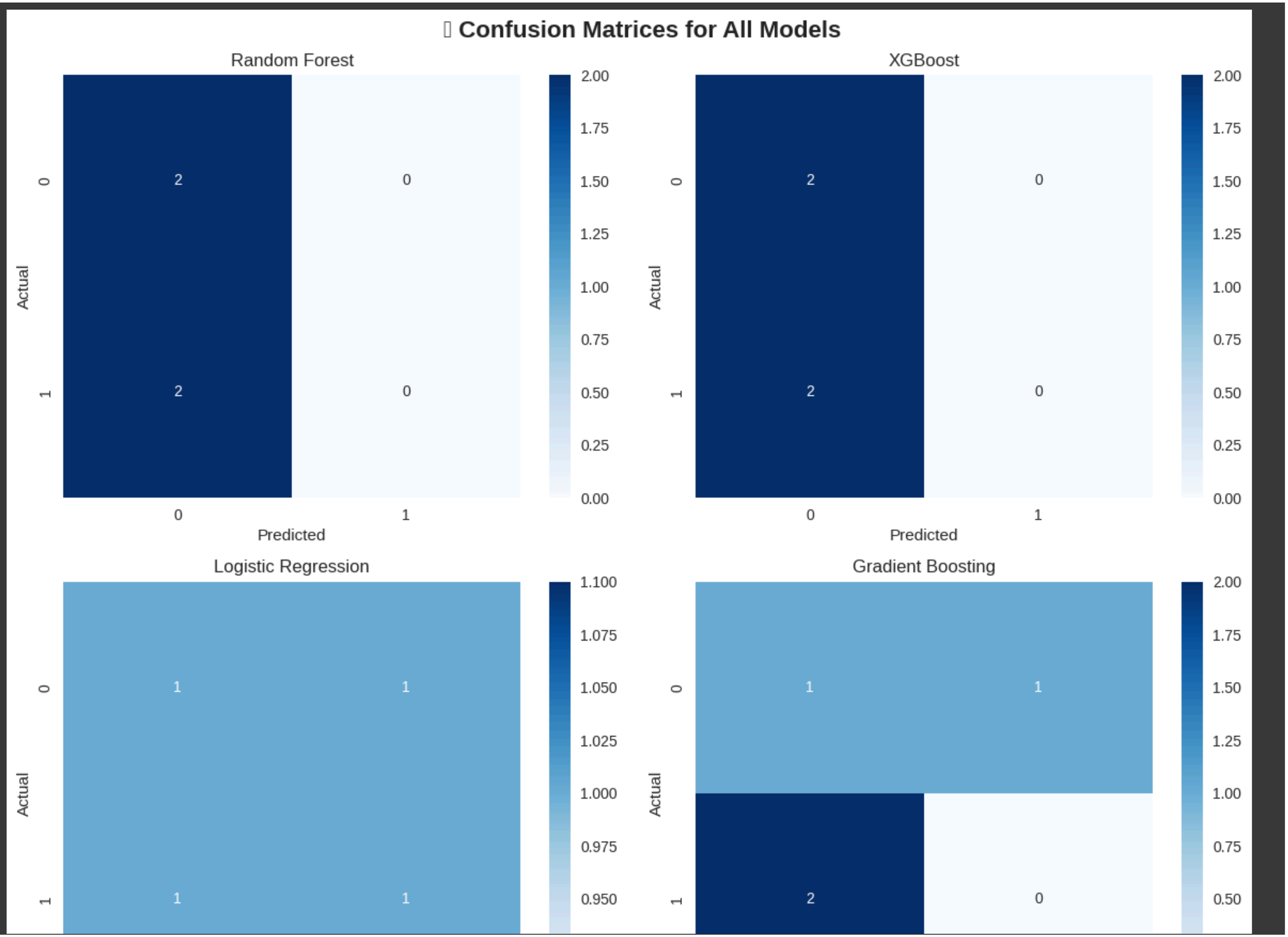
```
model_names = list(model_results.keys())
accuracies = [model_results[name]['accuracy'] for name in model_names]
cv_means = [model_results[name]['cv_mean'] for name in model_names]
cv_stds = [model_results[name]['cv_std'] for name in model_names]

fig = go.Figure()

fig.add_trace(go.Bar(
    name='Test Accuracy',
    x=model_names,
    y=accuracies,
    marker_color='lightblue'
))

fig.add_trace(go.Bar(
    name='CV Score',
    x=model_names,
    y=cv_means,
    error_y=dict(
        type='data',
        array=cv_stds,
        visible=True
    )
```

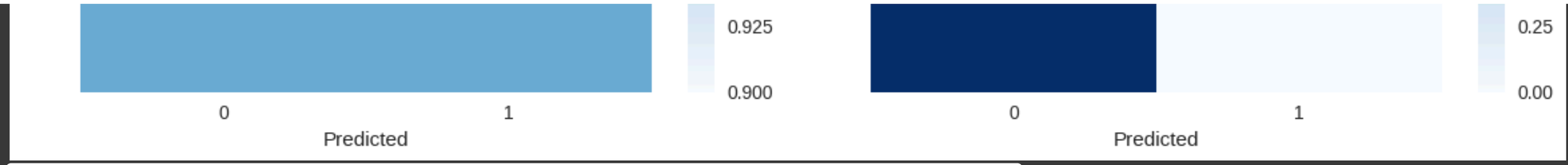
```
    ),
    marker_color='orange'
))

fig.update_layout(
    title='🎯 Model Performance Comparison',
    xaxis_title='Models',
    yaxis_title='Accuracy Score',
    barmode='group',
    height=500
)
fig.show()
```



Visual Evaluation of Classification Accuracy

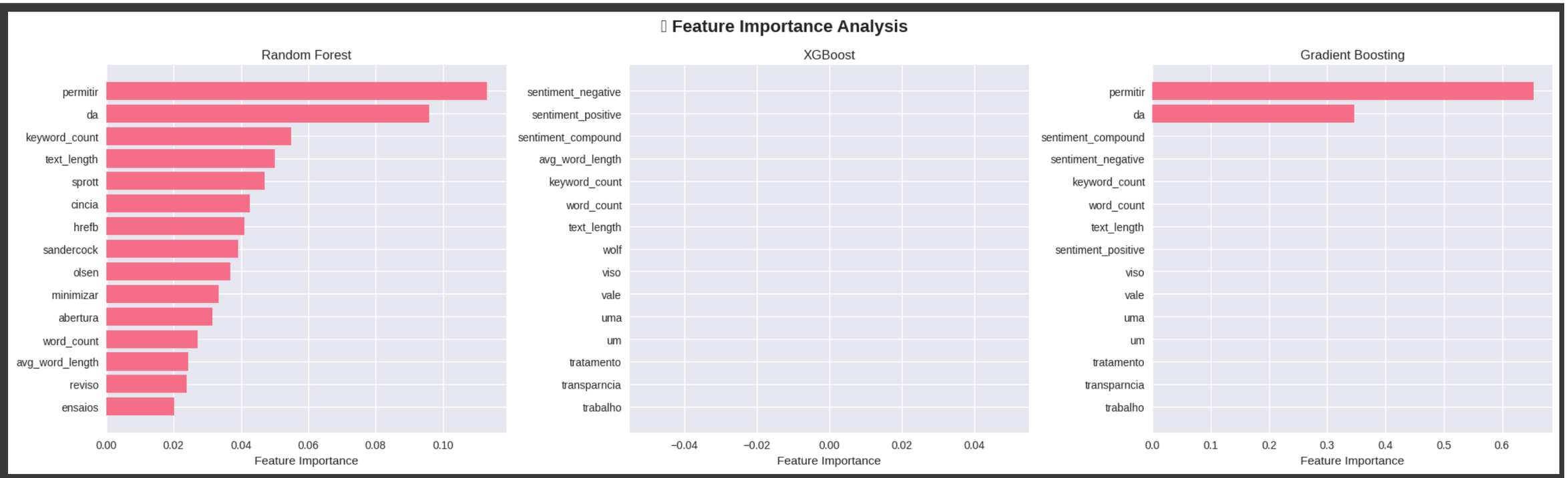
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix

num_models = len(model_results)
cols = 2
rows = (num_models + cols - 1) // cols

fig, axes = plt.subplots(rows, cols, figsize=(cols * 6, rows * 5))
fig.suptitle('🔄 Confusion Matrices for All Models', fontsize=16, fontweight='bold')
axes = axes.flatten()

for idx, (name, results) in enumerate(model_results.items()):
    y_true = results['test_labels']
    y_pred = results['predictions']
    labels = np.unique(y_true)

    cm = confusion_matrix(y_true, y_pred, labels=labels)
    sns.heatmap(cm, annot=True, fmt='d', ax=axes[idx],
                xticklabels=labels, yticklabels=labels,
                cmap='Blues')

    axes[idx].set_title(f'{name}', fontsize=12)
    axes[idx].set_xlabel('Predicted')
    axes[idx].set_ylabel('Actual')

for j in range(idx + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```


Confusion Matrices for All Models
Random Forest
Actual
Predicted
0
1
2
0
2
0
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
XGBoost
Logistic Regression
1
1
1.100
1.075
1.050
1.025
1.000
0.975
0.950
Gradient Boosting



Feature Importance Analysis (for tree-based models)

```
tree_models = ['Random Forest', 'XGBoost', 'Gradient Boosting']
fig, axes = plt.subplots(1, 3, figsize=(20, 6))
fig.suptitle('🌳 Feature Importance Analysis', fontsize=16, fontweight='bold')

for i, model_name in enumerate(tree_models):
    if model_name in model_results:
        model = model_results[model_name]['model']

        if hasattr(model, 'feature_importances_'):
            feature_importance = model.feature_importances_
            indices = np.argsort(feature_importance)[-15:]

            axes[i].barh(range(len(indices)), feature_importance[indices])
            axes[i].set_yticks(range(len(indices)))
            axes[i].set_yticklabels([ml_pipeline.feature_names[j] for j in indices])
            axes[i].set_title(f'{model_name}')
            axes[i].set_xlabel('Feature Importance')

plt.tight_layout()
plt.show()
```



ROC Curves (for binary classification adaptation)

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

print("📈 Generating ROC curves...")

best_model_name = max(model_results.keys(), key=lambda x: model_results[x]['accuracy'])
best_model = model_results[best_model_name]['model']

if hasattr(best_model, 'predict_proba'):
    X_test = model_results[best_model_name]['X_test']
    y_test = model_results[best_model_name]['test_labels']
    y_prob = best_model.predict_proba(X_test)

    n_classes = len(np.unique(y_test))
    y_test_bin = label_binarize(y_test, classes=range(n_classes))
```

```
    if n_classes > 2:
        fpr = dict()
        tpr = dict()
        roc_auc = dict()

        for i in range(n_classes):
            fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])
        plt.figure(figsize=(10, 8))
        colors = ['blue', 'red', 'green', 'orange', 'purple']

        for i, color in zip(range(n_classes), colors[:n_classes]):
            plt.plot(fpr[i], tpr[i], color=color, lw=2,
                     label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

        plt.plot([0, 1], [0, 1], 'k--', lw=2)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'📈 ROC Curves - {best_model_name}')
        plt.legend(loc="lower right")
        plt.grid(True, alpha=0.3)
        plt.show()
```

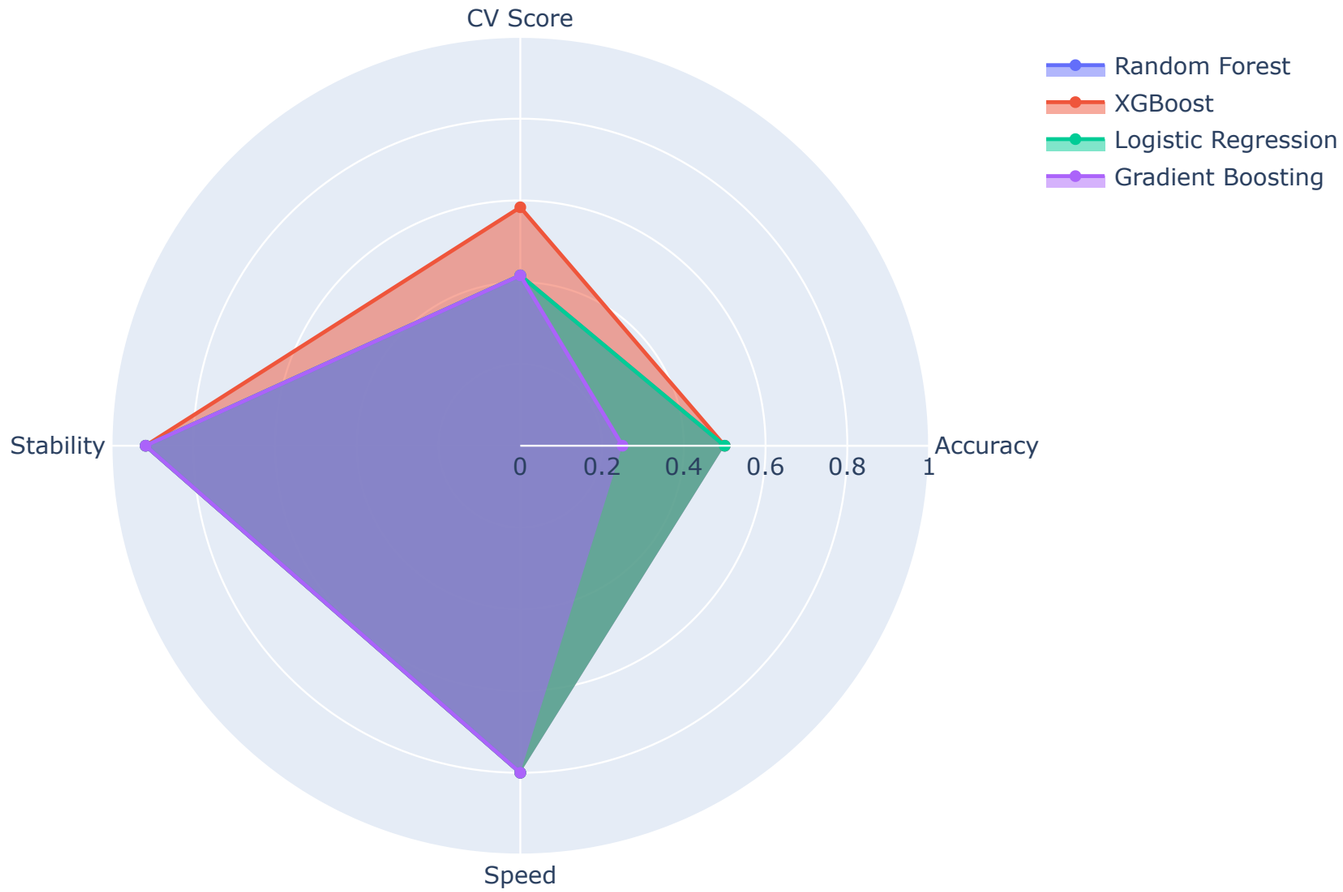
```
📈 Generating ROC curves...
```

Model Comparison Radar Chart

```
metrics_data = []
for name, results in model_results.items():
    metrics_data.append({
        'Model': name,
        'Accuracy': results['accuracy'],
        'CV_Mean': results['cv_mean'],
        'Stability': 1 - results['cv_std'],
        'Speed': 0.8
    })

metrics_df = pd.DataFrame(metrics_data)
fig = go.Figure()

for _, row in metrics_df.iterrows():
```

```
    fig.add_trace(go.Scatterpolar(
        r=[row['Accuracy'], row['CV_Mean'], row['Stability'], row['Speed']],
        theta=['Accuracy', 'CV Score', 'Stability', 'Speed'],
        fill='toself',
        name=row['Model']
    ))

fig.update_layout(
    polar=dict(
        radialaxis=dict(
            visible=True,
            range=[0, 1]
        )
    ),
    showlegend=True,
    title="🎯 Model Performance Radar Chart",
    height=600
)
fig.show()
print("✅ ML performance visualizations completed!")
```

🎯 Model Performance Radar Chart



✅ ML performance visualizations completed!

Deep Learning Models for Jira Ticket Classification

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, Dropout, Conv1D, GlobalMaxPooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

```
class DeepLearningPipeline:
    def __init__(self, max_features=5000, max_length=100):
        self.max_features = max_features
        self.max_length = max_length
        self.tokenizer = Tokenizer(num_words=max_features)
        self.models = {}

    def prepare_dl_data(self, texts, labels):
        """Prepare data for deep learning models"""
        self.tokenizer.fit_on_texts(texts)
        sequences = self.tokenizer.texts_to_sequences(texts)

        X = pad_sequences(sequences, maxlen=self.max_length)

        y = to_categorical(labels)

        return X, y

    def create_lstm_model(self, num_classes, embedding_dim=100):
        """Create LSTM model for text classification"""
        model = Sequential([
            Embedding(self.max_features, embedding_dim, input_length=self.max_length),
            LSTM(128, dropout=0.2, recurrent_dropout=0.2),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
        ])

        model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )

        return model

    def create_cnn_model(self, num_classes, embedding_dim=100):
        """Create CNN model for text classification"""
        model = Sequential([
            Embedding(self.max_features, embedding_dim, input_length=self.max_length),
            Conv1D(128, 5, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
```

```
        ])

        model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )

        return model

    def train_dl_models(self, X, y, validation_split=0.2, epochs=10, batch_size=32):
        """Train deep learning models"""
        num_classes = y.shape[1]

        models = {
            'LSTM': self.create_lstm_model(num_classes),
            'CNN': self.create_cnn_model(num_classes)
        }

        results = {}

        for name, model in models.items():
            print(f"🔄 Training {name} model...")

            history = model.fit(
                X, y,
                validation_split=validation_split,
                epochs=epochs,
                batch_size=batch_size,
                verbose=0
            )

            results[name] = {
                'model': model,
                'history': history,
                'final_accuracy': history.history['accuracy'][-1],
                'final_val_accuracy': history.history['val_accuracy'][-1]
            }

            print(f"✅ {name}: Accuracy = {history.history['accuracy'][-1]:.3f}, "
                  f"Val Accuracy = {history.history['val_accuracy'][-1]:.3f}")

        self.models = results
        return results
```

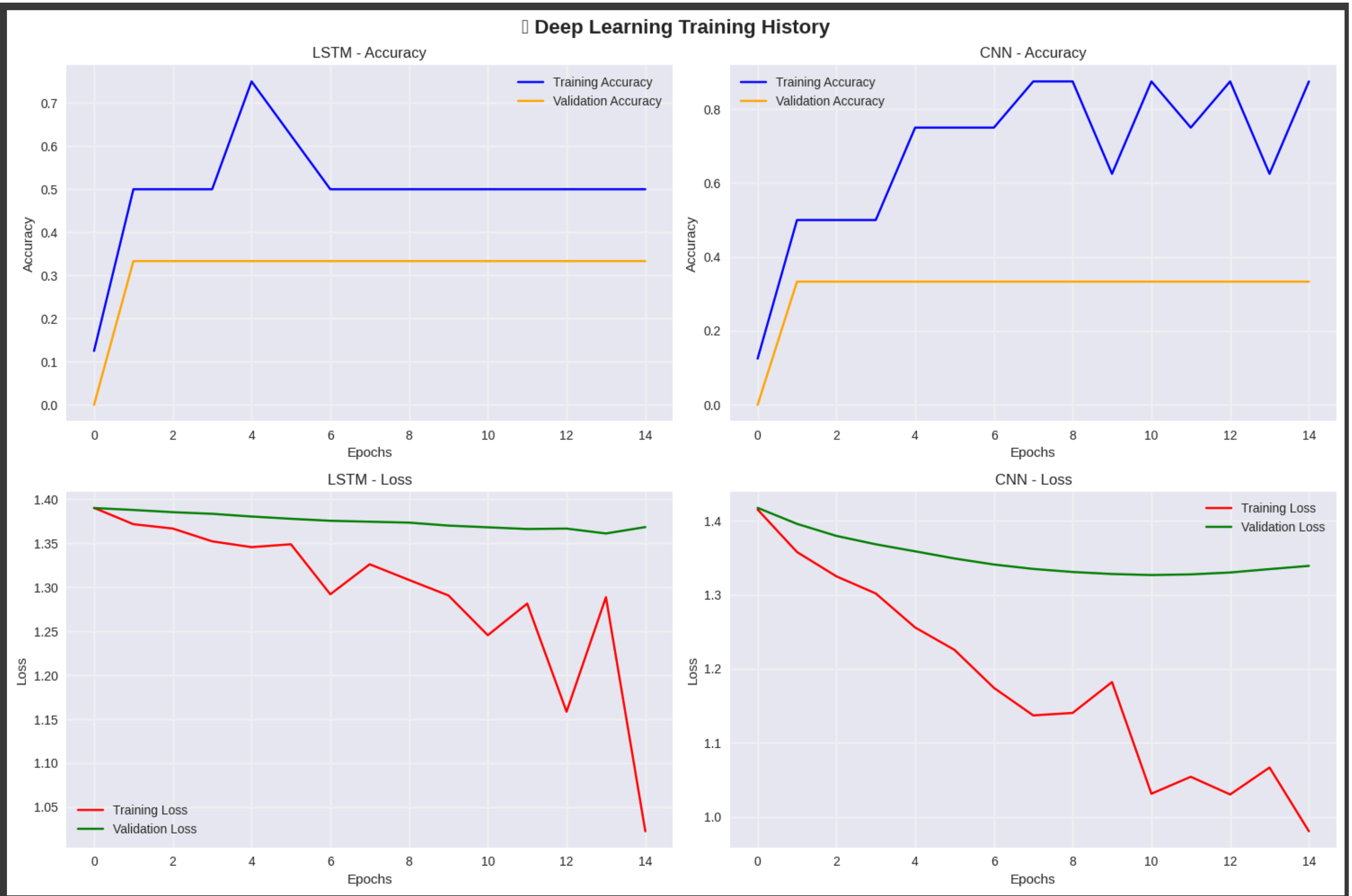
```
print("🔄 Preparing data for deep learning...")

dl_pipeline = DeepLearningPipeline()
X_dl, y_dl = dl_pipeline.prepare_dl_data(
    jira_processed['combined_text_clean'].fillna(''),
    jira_processed['priority_encoded']
)

print(f"📊 Deep learning data shape: X={X_dl.shape}, y={y_dl.shape}")

print("\n🧠 Training deep learning models...")
dl_results = dl_pipeline.train_dl_models(X_dl, y_dl, epochs=15)

print("\n✅ Deep learning pipeline completed!")
```

```
🔄 Preparing data for deep learning...
📊 Deep learning data shape: X=(11, 100), y=(11, 4)

🧠 Training deep learning models...
🔄 Training LSTM model...
✅ LSTM: Accuracy = 0.500, Val Accuracy = 0.333
🔄 Training CNN model...
✅ CNN: Accuracy = 0.875, Val Accuracy = 0.333

✅ Deep learning pipeline completed!
```

Deep Learning Training History

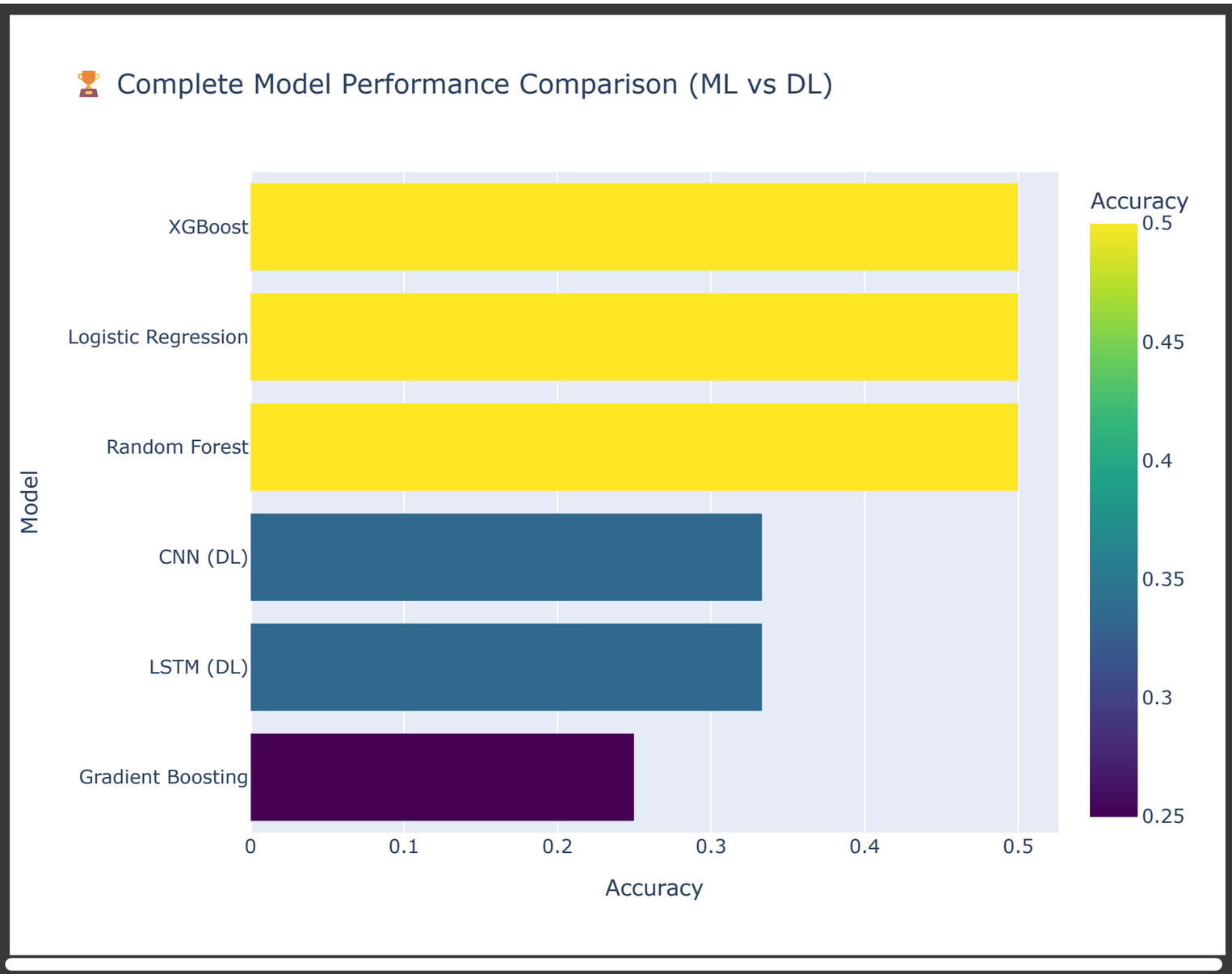
```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('🧠 Deep Learning Training History', fontsize=16, fontweight='bold')

for i, (name, results) in enumerate(dl_results.items()):
    history = results['history']

    axes[0, i].plot(history.history['accuracy'], label='Training Accuracy', color='blue')
    axes[0, i].plot(history.history['val_accuracy'], label='Validation Accuracy', color='orange')
    axes[0, i].set_title(f'{name} - Accuracy')
    axes[0, i].set_xlabel('Epochs')
    axes[0, i].set_ylabel('Accuracy')
    axes[0, i].legend()
    axes[0, i].grid(True, alpha=0.3)

    axes[1, i].plot(history.history['loss'], label='Training Loss', color='red')
```

```
    axes[1, i].plot(history.history['val_loss'], label='Validation Loss', color='green')
    axes[1, i].set_title(f'{name} - Loss')
    axes[1, i].set_xlabel('Epochs')
    axes[1, i].set_ylabel('Loss')
    axes[1, i].legend()
    axes[1, i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```


Deep Learning Training History
LSTM - Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
CNN - Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
LSTM - Loss
Training Loss
Validation Loss
Loss
Epochs
CNN - Loss
Training Loss
Validation Loss
Loss
Epochs

Model Performance Comparison (ML vs DL)

```
all_models = {}
all_models.update({name: results['accuracy'] for name, results in model_results.items()})
all_models.update({f"{name} (DL)": results['final_val_accuracy'] for name, results in dl_results.items()})

model_comparison_df = pd.DataFrame(list(all_models.items()), columns=['Model', 'Accuracy'])
model_comparison_df = model_comparison_df.sort_values('Accuracy', ascending=True)

fig = px.bar(model_comparison_df, x='Accuracy', y='Model', orientation='h',
             title='🏆 Complete Model Performance Comparison (ML vs DL)',
             color='Accuracy', color_continuous_scale='viridis')
fig.update_layout(height=600)
fig.show()
```

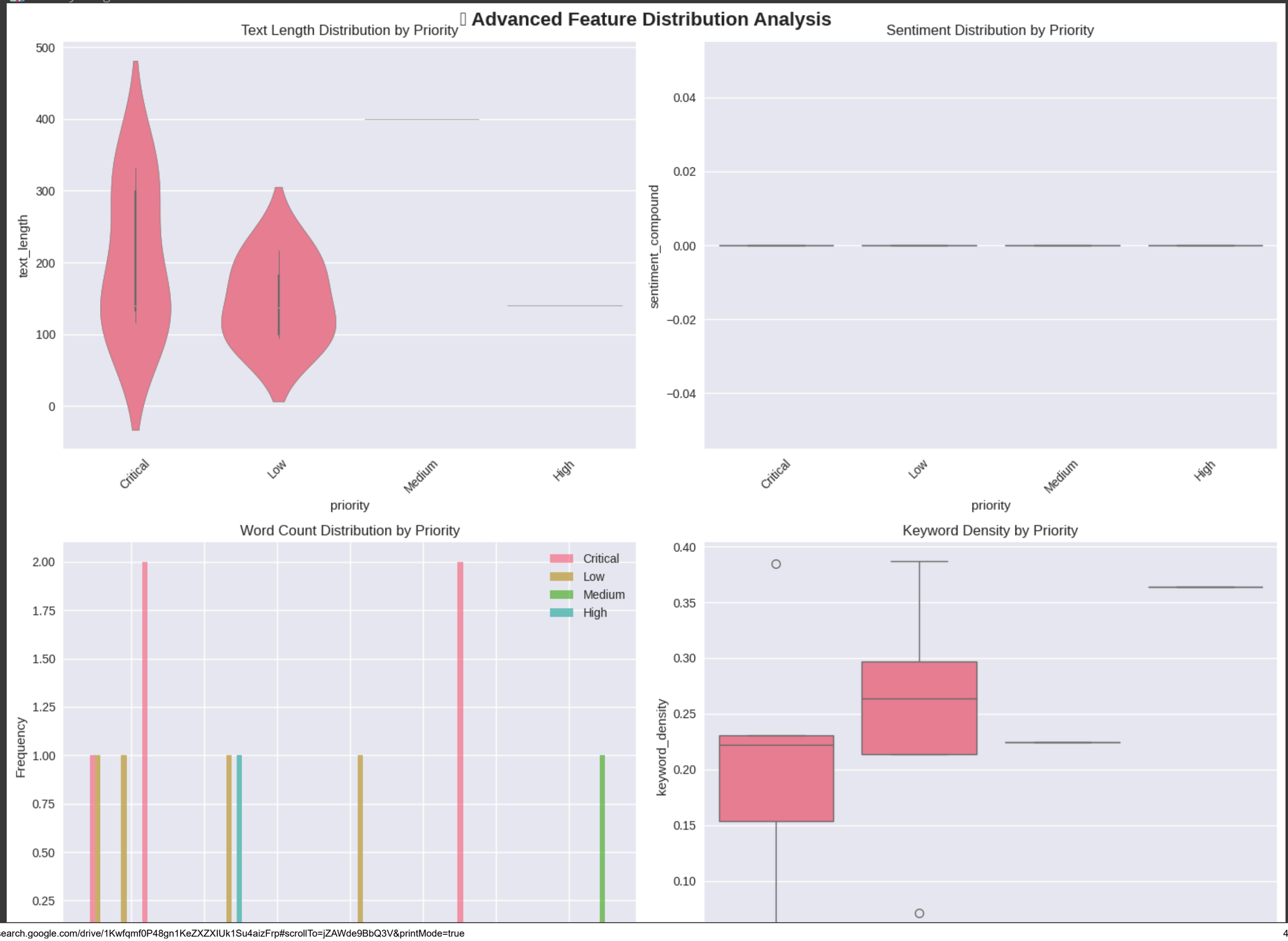


Feature Distribution Analysis

```
print("📊 Analyzing feature distributions...")

fig, axes = plt.subplots(3, 2, figsize=(15, 18))
fig.suptitle('📈 Advanced Feature Distribution Analysis', fontsize=16, fontweight='bold')

sns.violinplot(data=jira_processed, x=priority_col, y='text_length', ax=axes[0,0])
axes[0,0].set_title('Text Length Distribution by Priority')
axes[0,0].tick_params(axis='x', rotation=45)
```

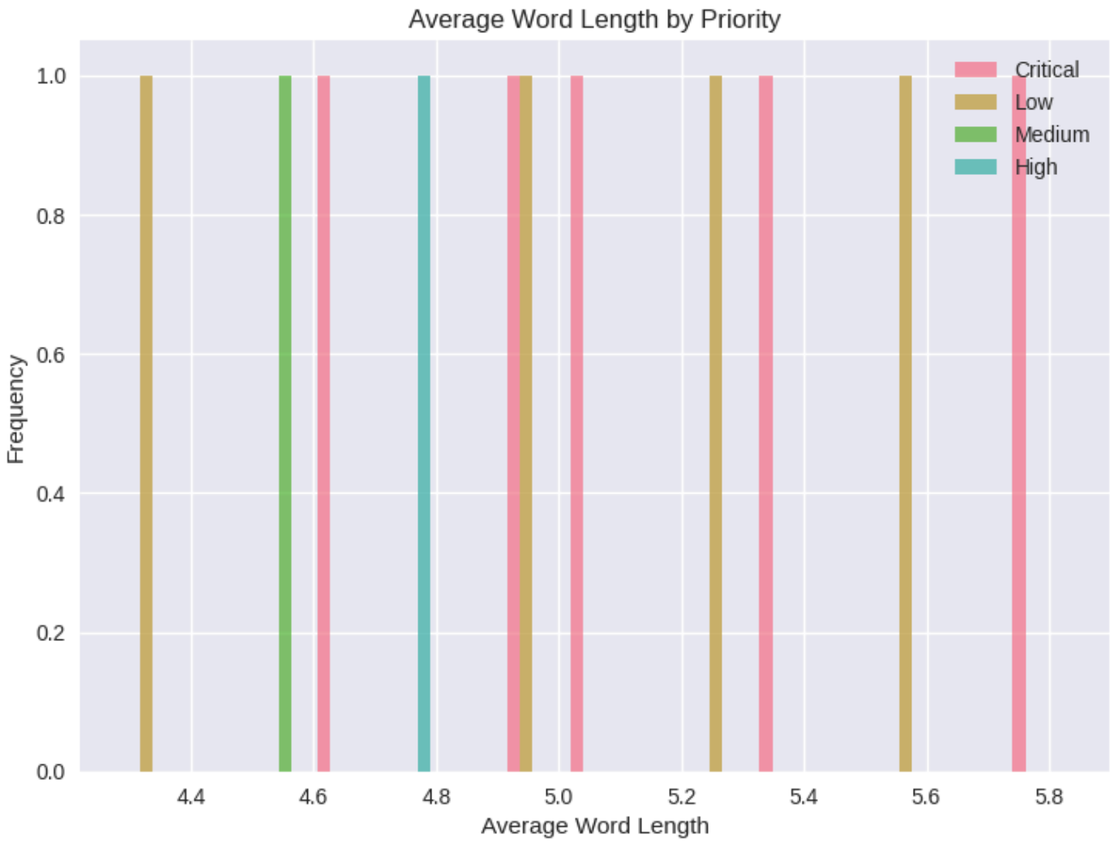
```
sns.boxplot(data=jira_processed, x=priority_col, y='sentiment_compound', ax=axes[0,1])
axes[0,1].set_title('Sentiment Distribution by Priority')
axes[0,1].tick_params(axis='x', rotation=45)

axes[1,0].hist([jira_processed[jira_processed[priority_col] == p]['word_count'] for p in jira_processed[priority_col].unique()],
               bins=20, alpha=0.7, label=jira_processed[priority_col].unique())
axes[1,0].set_title('Word Count Distribution by Priority')
axes[1,0].set_xlabel('Word Count')
axes[1,0].set_ylabel('Frequency')
axes[1,0].legend()

sns.boxplot(data=jira_processed, x=priority_col, y='keyword_density', ax=axes[1,1])
axes[1,1].set_title('Keyword Density by Priority')
axes[1,1].tick_params(axis='x', rotation=45)

axes[2,0].hist([jira_processed[jira_processed[priority_col] == p]['avg_word_length'] for p in jira_processed[priority_col].unique()],
               bins=15, alpha=0.7, label=jira_processed[priority_col].unique())
axes[2,0].set_title('Average Word Length by Priority')
axes[2,0].set_xlabel('Average Word Length')
axes[2,0].set_ylabel('Frequency')
axes[2,0].legend()

sentiment_by_priority = jira_processed.groupby(priority_col)[
    ['sentiment_positive', 'sentiment_negative', 'sentiment_neutral']
].mean()

sentiment_by_priority.plot(kind='bar', stacked=True, ax=axes[2,1],
                           color=['green', 'red', 'gray'], alpha=0.7)
axes[2,1].set_title('Average Sentiment Components by Priority')
axes[2,1].set_xlabel('Priority')
axes[2,1].set_ylabel('Sentiment Score')
axes[2,1].tick_params(axis='x', rotation=45)
axes[2,1].legend()

plt.tight_layout()
plt.show()
```

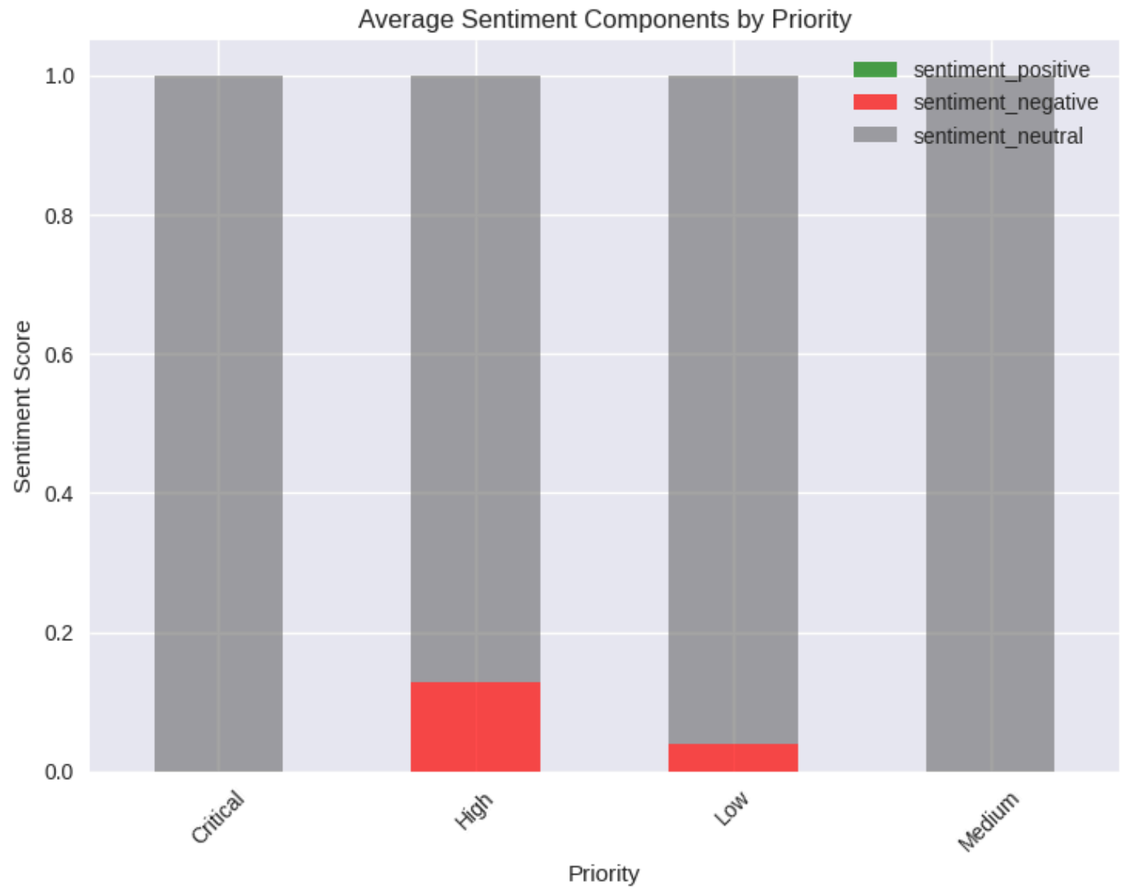
```
Analyzing feature distributions...
```




0.00
15
20
25
30
35
40
45
Word Count


0.05
Critical
Low
Medium
High
priority


Average Word Length by Priority
Critical
Low
Medium
High
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
4.4
4.6
4.8
5.0
5.2
5.4
5.6
5.8
Average Word Length

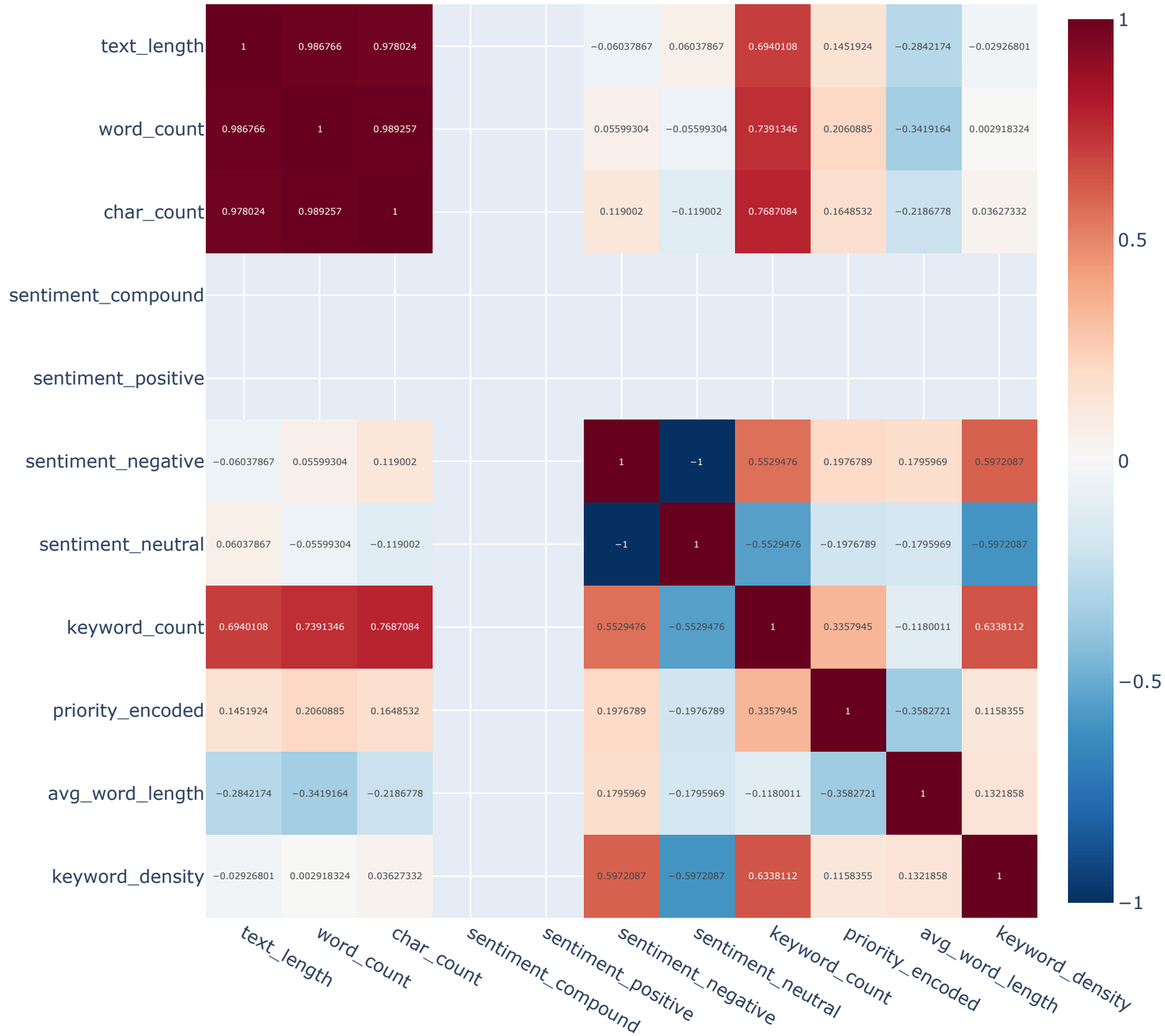

Average Sentiment Components by Priority
sentiment_positive
sentiment_negative
sentiment_neutral
Sentiment Score
1.0
0.8
0.6
0.4
0.2
0.0
Critical
High
Low
Medium
Priority

Advanced Correlation Analysis

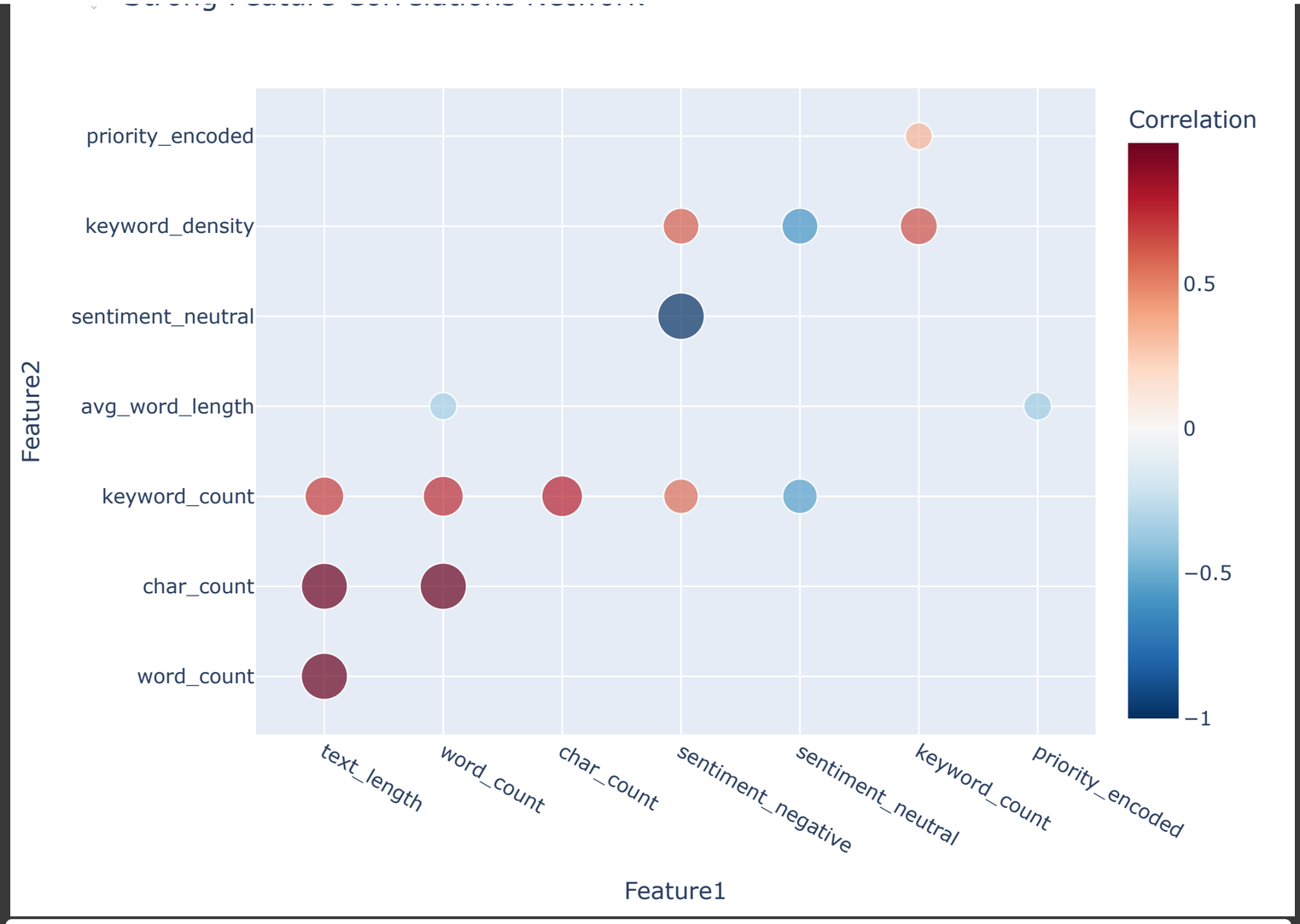
```
 print("🔗 Creating advanced correlation analysis...")

numerical_features = jira_processed.select_dtypes(include=[np.number]).columns
correlation_matrix = jira_processed[numerical_features].corr()

fig = px.imshow(correlation_matrix,
                text_auto=True,
                aspect="auto",
                color_continuous_scale='RdBu_r',
                title="🔗 Complete Feature Correlation Matrix")
fig.update_layout(height=800, width=800)
fig.show()

strong_correlations = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        corr_val = correlation_matrix.iloc[i, j]
        if abs(corr_val) > 0.3:
            strong_correlations.append({
                'Feature1': correlation_matrix.columns[i],
                'Feature2': correlation_matrix.columns[j],
                'Correlation': corr_val
            })

if strong_correlations:
    corr_df = pd.DataFrame(strong_correlations)

    fig = px.scatter(corr_df, x='Feature1', y='Feature2',
                     size=abs(corr_df['Correlation']),
                     color='Correlation',
                     color_continuous_scale='RdBu_r',
                     title=' 🔗 Strong Feature Correlations Network')
    fig.update_layout(height=600)
    fig.show()
```

Creating advanced correlation analysis...

Complete Feature Correlation Matrix



Strong Feature Correlations Network


priority_encoded
keyword_density
sentiment_neutral
avg_word_length
keyword_count
char_count
word_count
Feature2
text_length
word_count
char_count
sentiment_negative
sentiment_neutral
keyword_count
priority_encoded
Feature1
Correlation
0.5
0
−0.5
−1

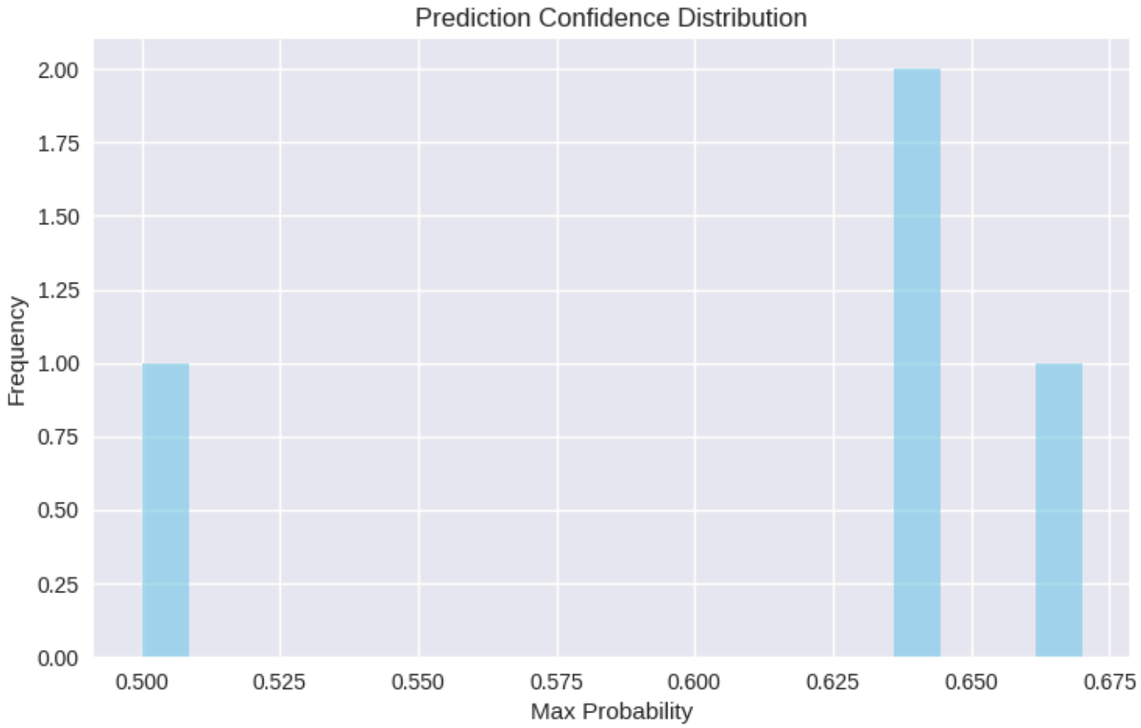
Priority Prediction Confidence Analysis

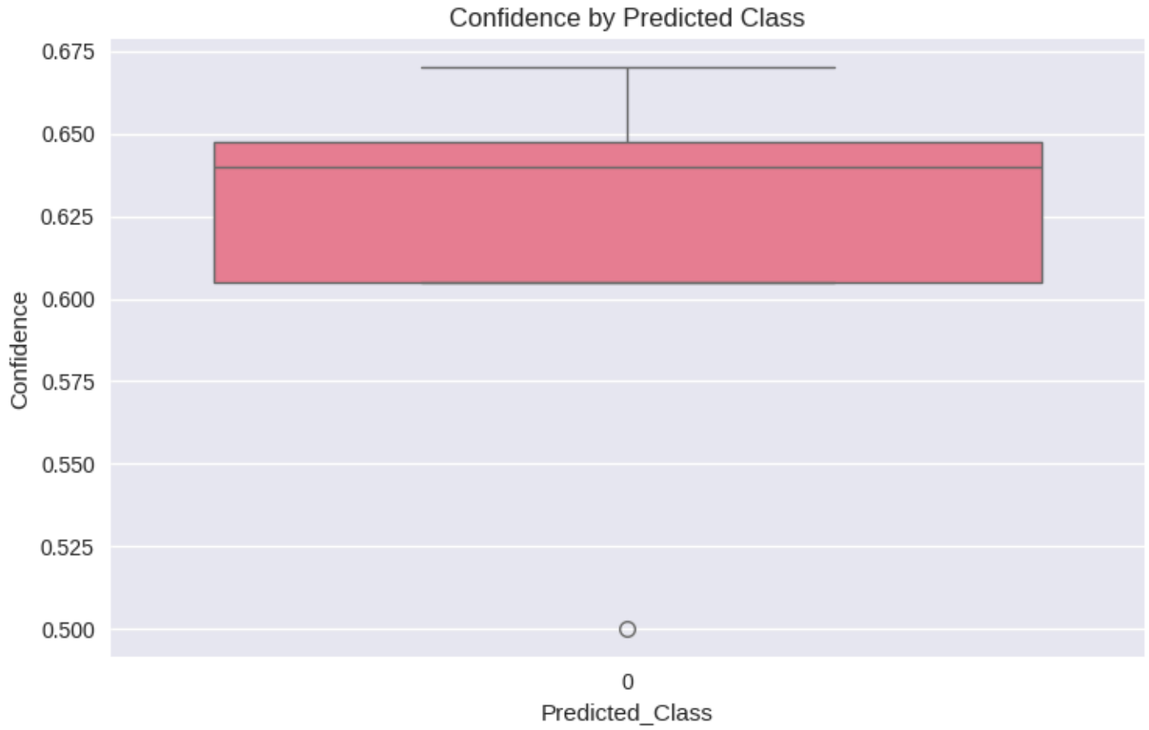
```
print("🎯 Analyzing prediction confidence...")

best_model_name = max(model_results.keys(), key=lambda x: model_results[x]['accuracy'])
best_model = model_results[best_model_name]['model']

if hasattr(best_model, 'predict_proba'):
    X_sample = model_results[best_model_name]['X_test'][:100]
    probabilities = best_model.predict_proba(X_sample)

    max_probs = np.max(probabilities, axis=1)
    predicted_classes = np.argmax(probabilities, axis=1)

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    fig.suptitle('🎯 Prediction Confidence Analysis', fontsize=16, fontweight='bold')

    axes[0,0].hist(max_probs, bins=20, alpha=0.7, color='skyblue')
    axes[0,0].set_title('Prediction Confidence Distribution')
    axes[0,0].set_xlabel('Max Probability')
    axes[0,0].set_ylabel('Frequency')

    conf_by_class = pd.DataFrame({
        'Predicted_Class': predicted_classes,
        'Confidence': max_probs
    })

    sns.boxplot(data=conf_by_class, x='Predicted_Class', y='Confidence', ax=axes[0,1])
    axes[0,1].set_title('Confidence by Predicted Class')

    sample_probs = probabilities[:20]
    sns.heatmap(sample_probs.T, annot=True, fmt='.2f', ax=axes[1,0],
                cmap='YlOrRd', xticklabels=range(20),
                yticklabels=[f'Class {i}' for i in range(sample_probs.shape[1])])
    axes[1,0].set_title('Prediction Probabilities Heatmap (Sample)')
    axes[1,0].set_xlabel('Sample Index')

    high_conf_mask = max_probs > 0.8
    low_conf_mask = max_probs < 0.6

    conf_analysis = pd.DataFrame({
        'Confidence_Level': ['High (>0.8)', 'Medium (0.6-0.8)', 'Low (<0.6)'],
```

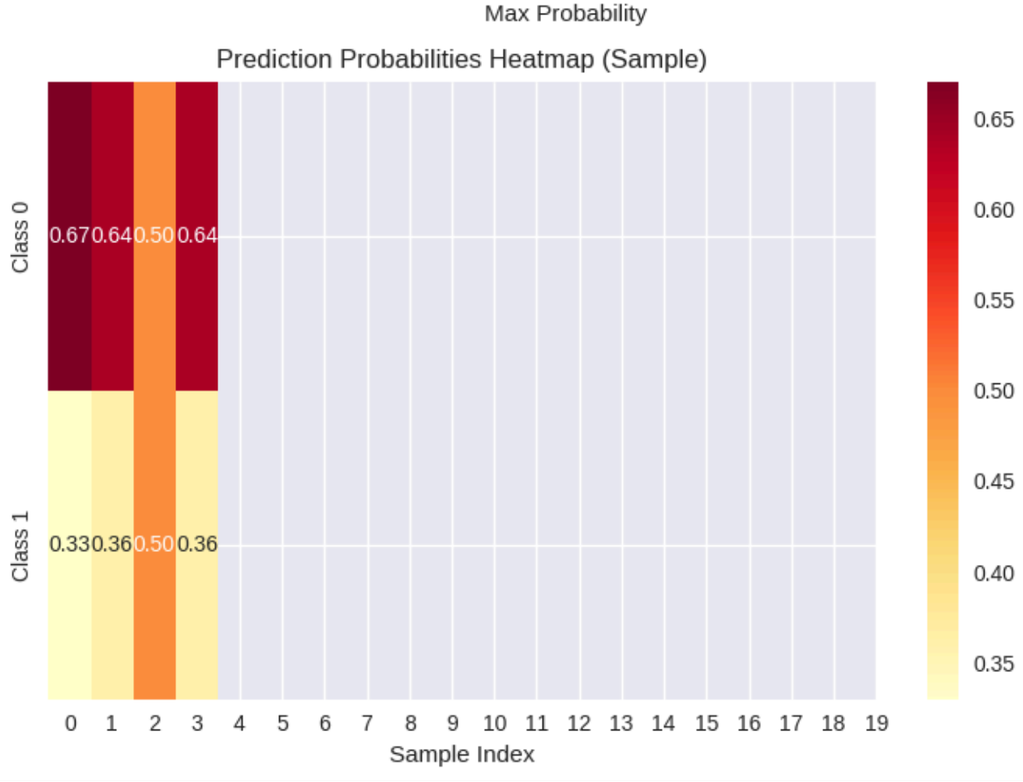
```
        'Count': [
            np.sum(high_conf_mask),
            np.sum((max_probs >= 0.6) & (max_probs <= 0.8)),
            np.sum(low_conf_mask)
        ]
    })

    axes[1,1].pie(conf_analysis['Count'], labels=conf_analysis['Confidence_Level'],
                  autopct='%1.1f%%', startangle=90)
    axes[1,1].set_title('Prediction Confidence Distribution')

    plt.tight_layout()
    plt.show()

print("✅ Advanced analytics visualizations completed!")
```

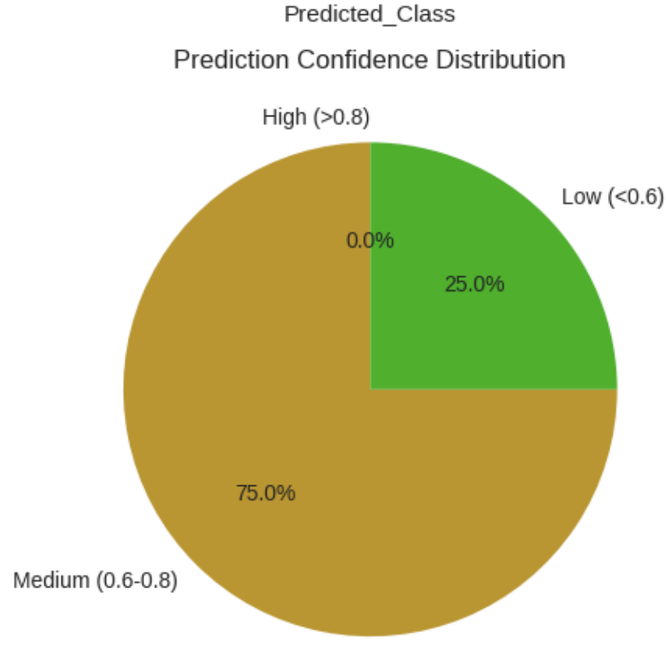
```
🎯 Analyzing prediction confidence...
```

Prediction Confidence Analysis







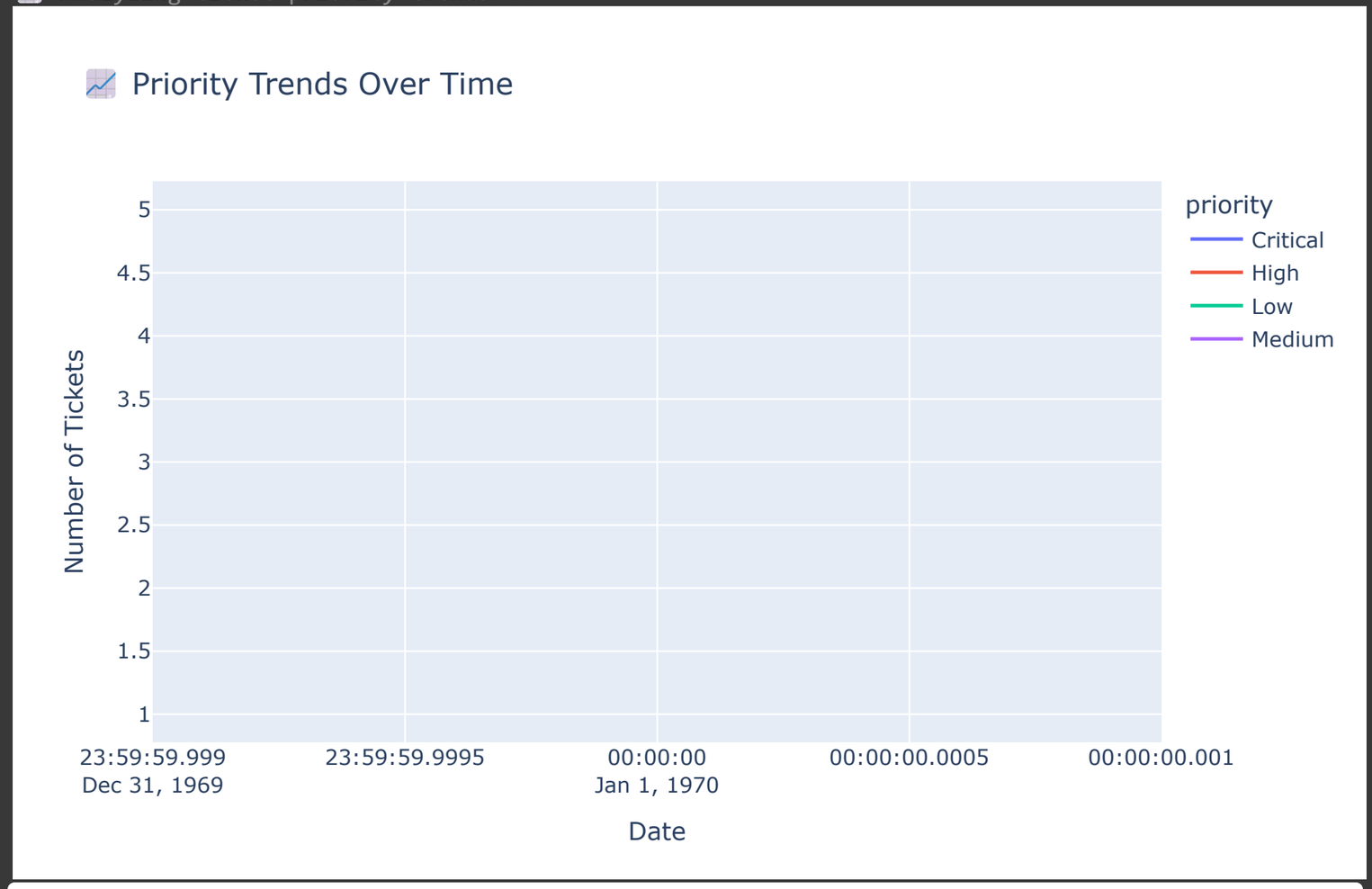



```
✅ Advanced analytics visualizations completed!
```

Ticket Priority Trends Analysis

```
print("📈 Analyzing ticket priority trends...")

date_cols = [col for col in jira_processed.columns if 'date' in col.lower() or 'time' in col.lower()]

if date_cols:
    date_col = date_cols[0]
    jira_processed[date_col] = pd.to_datetime(jira_processed[date_col], errors='coerce')

    priority_trends = jira_processed.groupby([
        jira_processed[date_col].dt.date, priority_col
    ]).size().unstack(fill_value=0)

    fig = px.line(priority_trends, title='📈 Priority Trends Over Time')
    fig.update_layout(height=500, xaxis_title='Date', yaxis_title='Number of Tickets')
    fig.show()
```

```
Analyzing ticket priority trends...
```

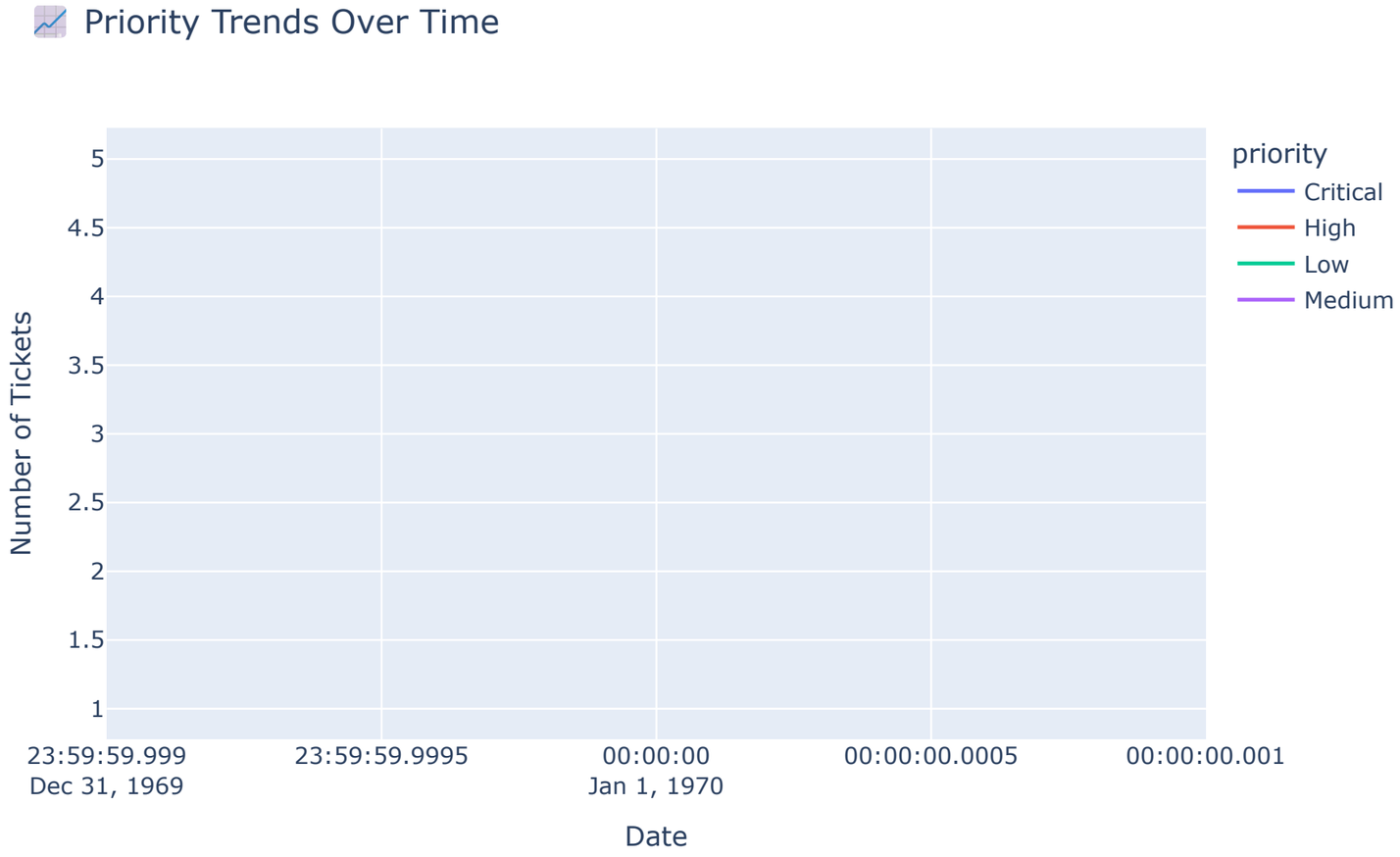


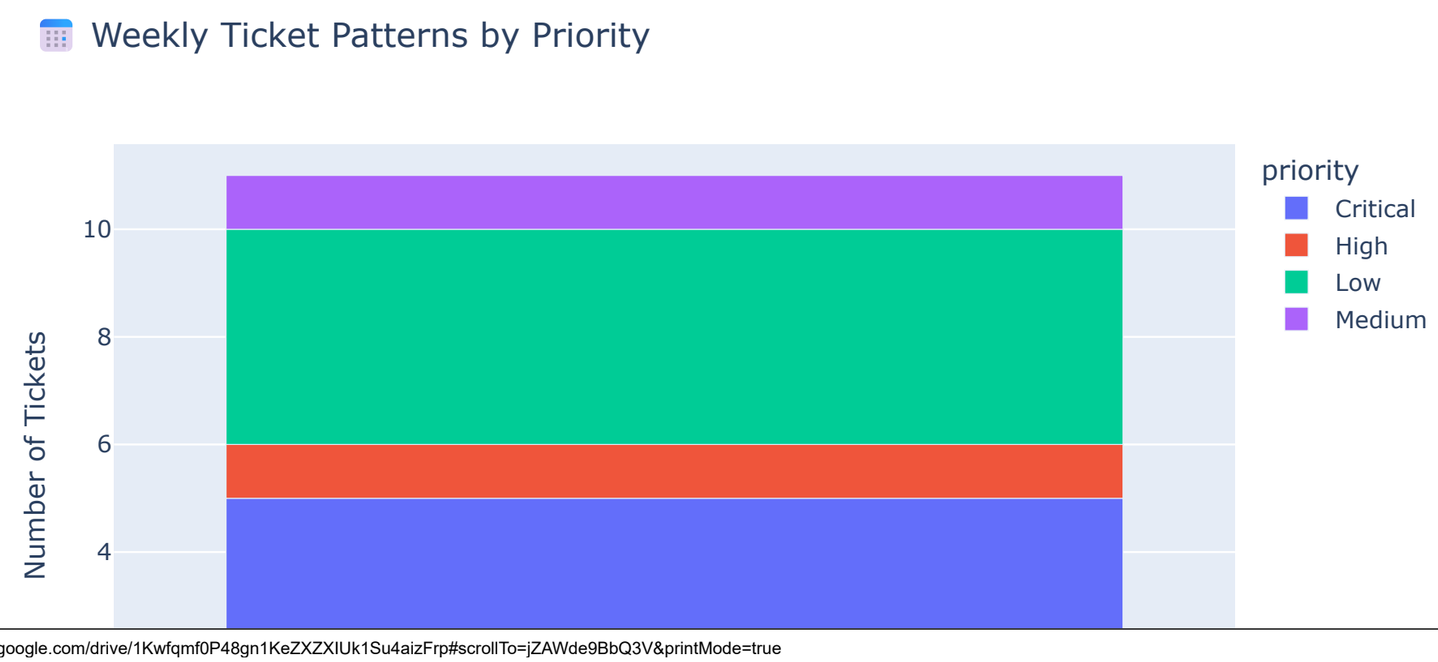
Ticket Priority Trends Analysis

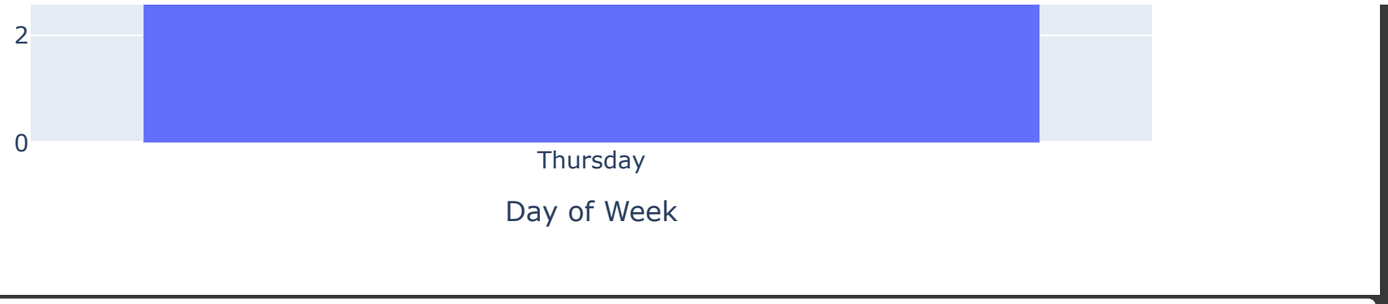
```
print("📈 Analyzing ticket priority trends...")

date_cols = [col for col in jira_processed.columns if 'date' in col.lower() or 'time' in col.lower()]

if date_cols:
    date_col = date_cols[0]
    jira_processed[date_col] = pd.to_datetime(jira_processed[date_col], errors='coerce')

    priority_trends = jira_processed.groupby([
        jira_processed[date_col].dt.date, priority_col
    ]).size().unstack(fill_value=0)

    fig = px.line(priority_trends, title='📈 Priority Trends Over Time')
```

```
    fig = px.line(priority_trends, title='📈 Priority Trends Over Time')
    fig.update_layout(height=500, xaxis_title='Date', yaxis_title='Number of Tickets')
    fig.show()
    jira_processed['day_of_week'] = jira_processed[date_col].dt.day_name()
    weekly_pattern = jira_processed.groupby(['day_of_week', priority_col]).size().unstack(fill_value=0)

    fig = px.bar(weekly_pattern, title='📅 Weekly Ticket Patterns by Priority')
    fig.update_layout(height=500, xaxis_title='Day of Week', yaxis_title='Number of Tickets')
    fig.show()

else:
    print("📅 Creating synthetic time analysis...")

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    fig.suptitle('📈 Priority Distribution Analysis', fontsize=16, fontweight='bold')

    priority_counts = jira_processed[priority_col].value_counts()
    axes[0,0].pie(priority_counts.values, labels=priority_counts.index, autopct='%1.1f%%')
    axes[0,0].set_title('Overall Priority Distribution')

    sns.boxplot(data=jira_processed, x=priority_col, y='sentiment_compound', ax=axes[0,1])
    axes[0,1].set_title('Sentiment by Priority Level')
    axes[0,1].tick_params(axis='x', rotation=45)

    sns.boxplot(data=jira_processed, x=priority_col, y='word_count', ax=axes[1,0])
    axes[1,0].set_title('Text Complexity by Priority')
    axes[1,0].tick_params(axis='x', rotation=45)

    sns.boxplot(data=jira_processed, x=priority_col, y='keyword_density', ax=axes[1,1])
    axes[1,1].set_title('Keyword Density by Priority')
    axes[1,1].tick_params(axis='x', rotation=45)

    plt.tight_layout()
    plt.show()
```

```
Analyzing ticket priority trends...
```







Component Analysis (if component column exists)

```
component_cols = [col for col in jira_processed.columns if 'component' in col.lower()]

if component_cols:
    component_col = component_cols[0]

    component_priority = pd.crosstab(jira_processed[component_col], jira_processed[priority_col])

    fig = px.imshow(component_priority,
                    text_auto=True,
                    title='🔧 Component vs Priority Heatmap',
                    color_continuous_scale='Blues')
    fig.update_layout(height=500)
    fig.show()

    component_sentiment = jira_processed.groupby(component_col)['sentiment_compound'].mean().sort_values(ascending=False)

    fig = px.bar(x=component_sentiment.values, y=component_sentiment.index,
                 orientation='h', title='😊 Average Sentiment by Component')
    fig.update_layout(height=400)
    fig.show()
```

Advanced Text Analytics Dashboard

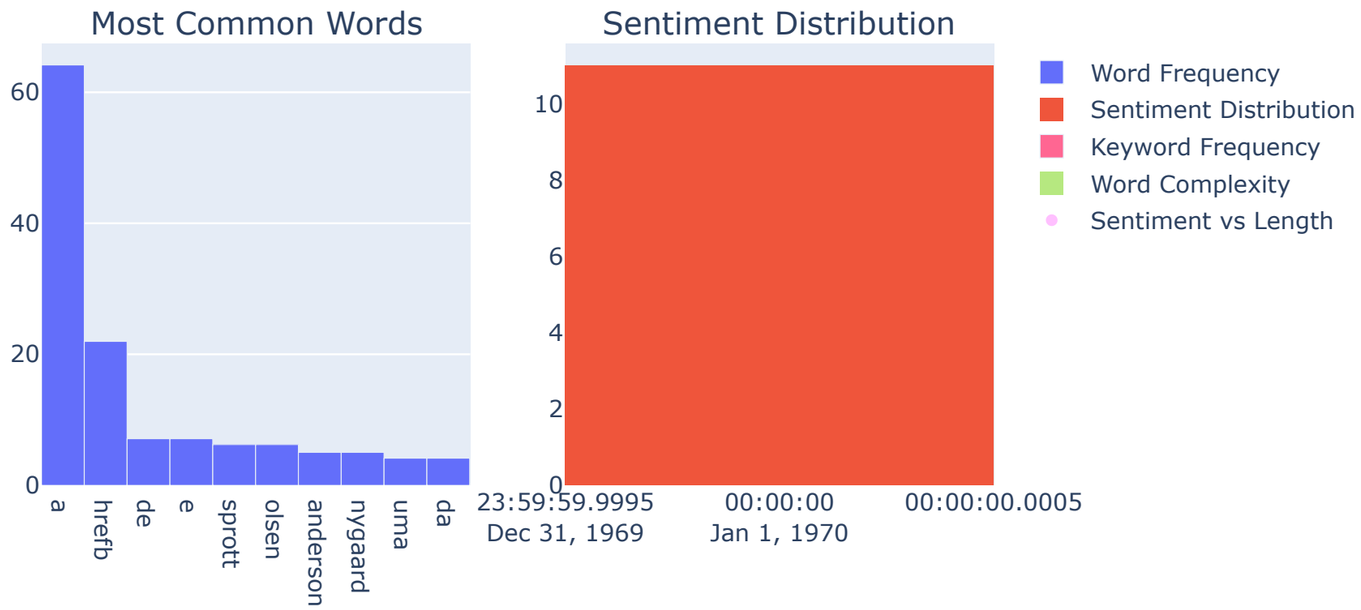
```
print("📝 Creating advanced text analytics dashboard...")

fig = make_subplots(
    rows=3, cols=2,
    subplot_titles=('Most Common Words', 'Sentiment Distribution',
                    'Text Length vs Priority', 'Keyword Analysis',
```

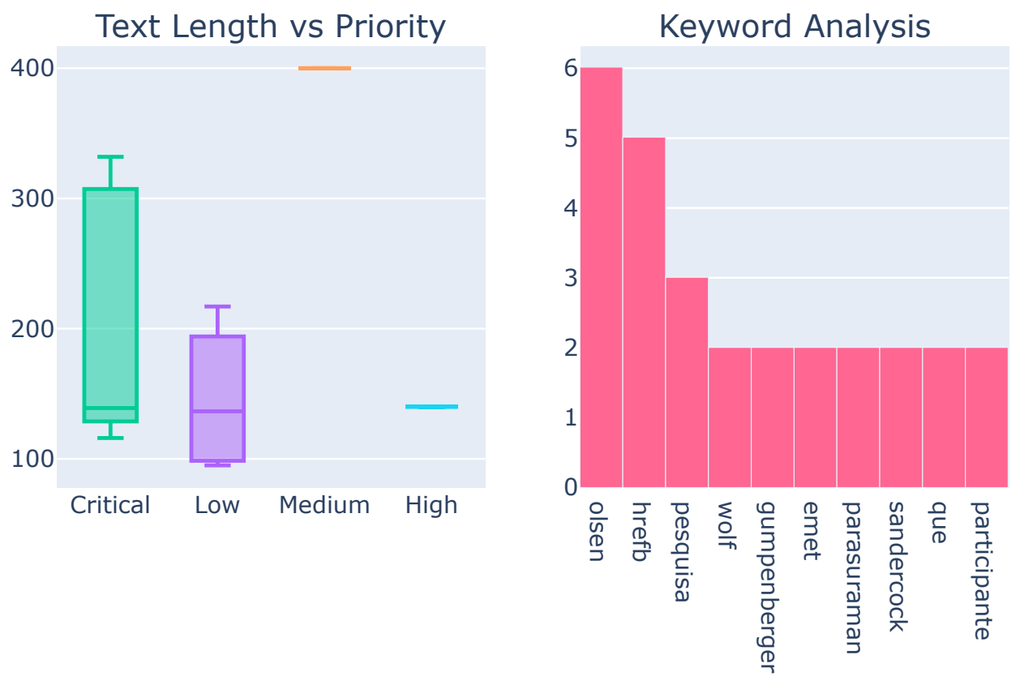
```
                    'Word Complexity', 'Sentiment vs Text Length'),
    specs=[[{"type": "bar"}, {"type": "histogram"}],
           [{"type": "box"}, {"type": "bar"}],
           [{"type": "histogram"}, {"type": "scatter"}]]
)

all_text = ' '.join(jira_processed['combined_text_clean'].dropna())
word_freq = Counter(all_text.split())
common_words = word_freq.most_common(10)

fig.add_trace(go.Bar(
    x=[word for word, count in common_words],
    y=[count for word, count in common_words],
    name="Word Frequency"
), row=1, col=1)

fig.add_trace(go.Histogram(
    x=jira_processed['sentiment_compound'],
    nbinsx=30,
    name="Sentiment Distribution"
), row=1, col=2)

for priority in jira_processed[priority_col].unique():
    priority_data = jira_processed[jira_processed[priority_col] == priority]
    fig.add_trace(go.Box(
        y=priority_data['text_length'],
        name=priority,
        showlegend=False
    ), row=2, col=1)

if top_keywords:
    top_10_keywords = top_keywords[:10]
    fig.add_trace(go.Bar(
        x=[keyword for keyword, count in top_10_keywords],
        y=[count for keyword, count in top_10_keywords],
        name="Keyword Frequency"
    ), row=2, col=2)

fig.add_trace(go.Histogram(
    x=jira_processed['avg_word_length'],
    nbinsx=25,
    name="Word Complexity"
), row=3, col=1)

fig.add_trace(go.Scatter(
    x=jira_processed['text_length'],
```

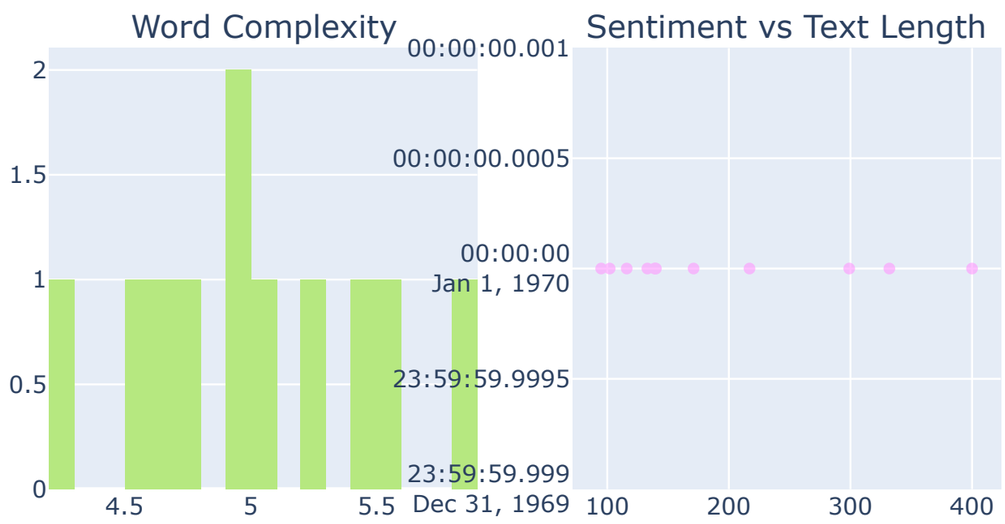
```
    y=jira_processed['sentiment_compound'],
    mode='markers',
    name="Sentiment vs Length",
    opacity=0.6
), row=3, col=2)

fig.update_layout(height=1200, title_text="📝 Advanced Text Analytics Dashboard")
fig.show()
```

```
Creating advanced text analytics dashboard...
```

Advanced Text Analytics Dashboard





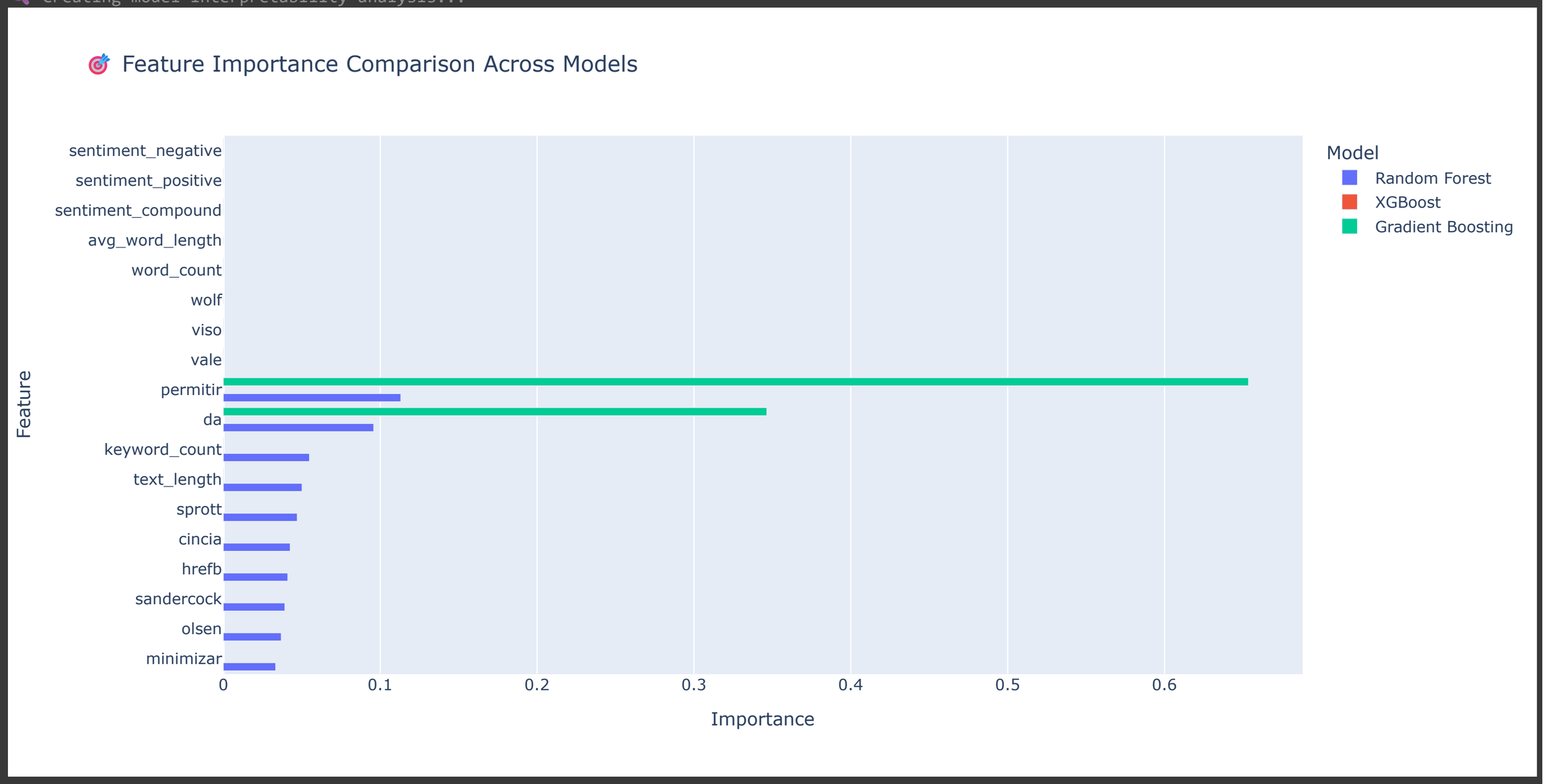


Interpreting Model Decisions: Top Features by Importance

```
print("🔍 Creating model interpretability analysis...")

feature_importance_data = []

for model_name in ['Random Forest', 'XGBoost', 'Gradient Boosting']:
    if model_name in model_results:
        model = model_results[model_name]['model']
        if hasattr(model, 'feature_importances_'):
            importance = model.feature_importances_
            indices = np.argsort(importance)[-10:]

            for idx in indices:
                feature_importance_data.append({
                    'Model': model_name,
                    'Feature': ml_pipeline.feature_names[idx],
                    'Importance': importance[idx]
                })

if feature_importance_data:
    importance_df = pd.DataFrame(feature_importance_data)

    fig = px.bar(importance_df, x='Importance', y='Feature', color='Model'
```

```
fig = px.bar(importance_df, x='Importance', y='Feature', color='Model',
             orientation='h', title='🎯 Feature Importance Comparison Across Models',
             barmode='group')
fig.update_layout(height=600)
fig.show()
```

```
🔍 Creating model interpretability analysis...
```



Business Impact Dashboard

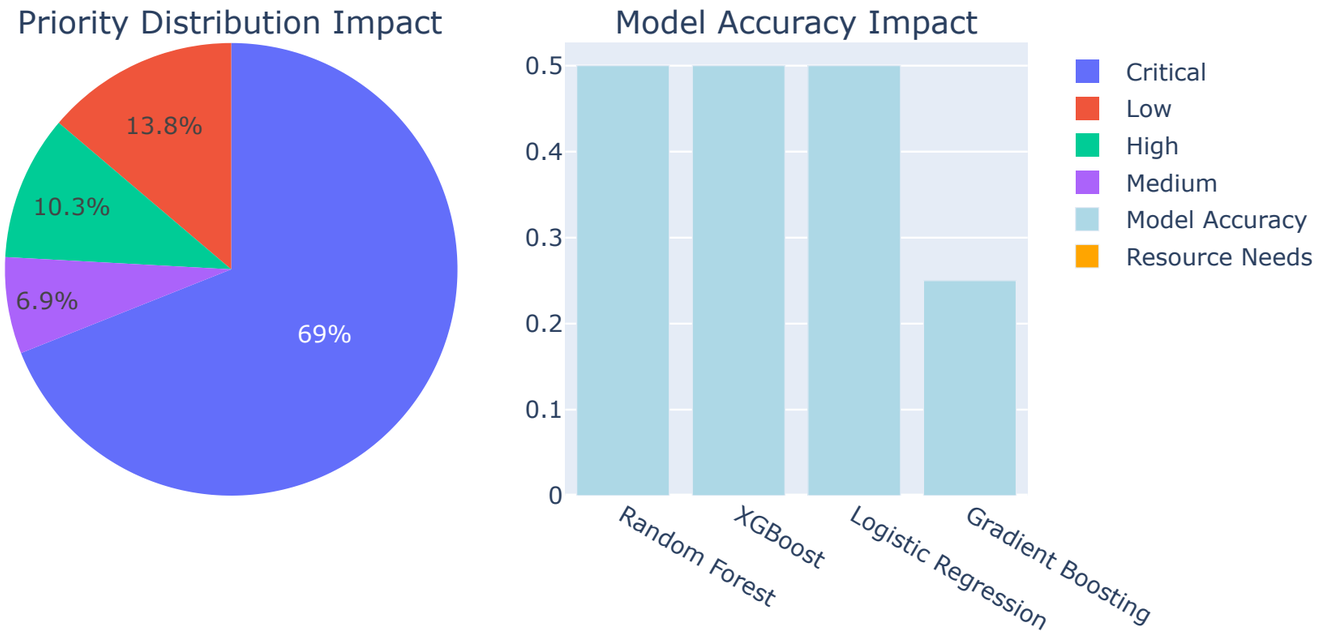
```
print("💼 Creating business impact dashboard...")
```

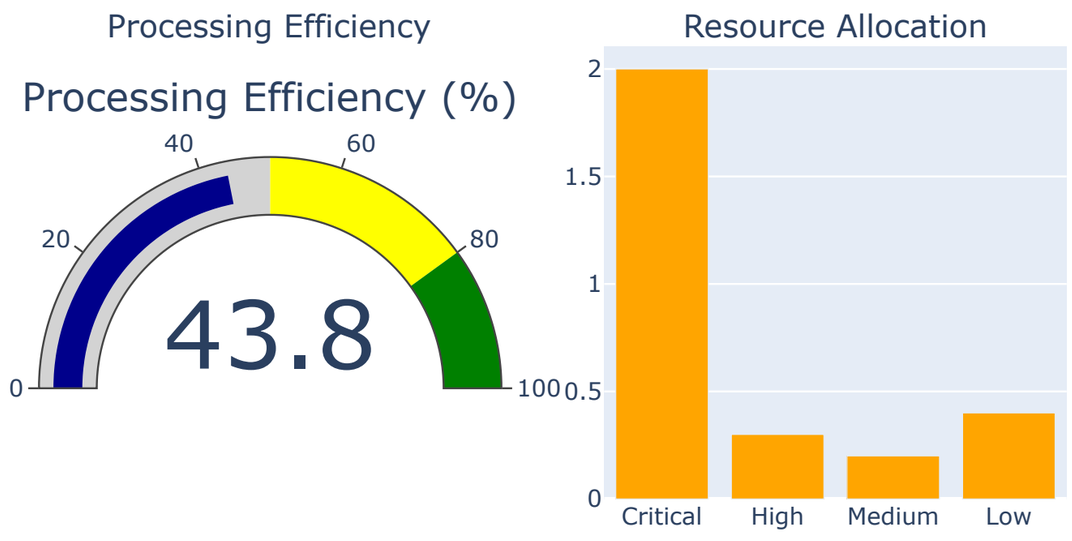
```
priority_counts = jira_processed[priority_col].value_counts().to_dict()

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Priority Distribution Impact', 'Model Accuracy Impact',
                    'Processing Efficiency', 'Resource Allocation'),
    specs=[[{"type": "pie"}, {"type": "bar"}],
           [{"type": "indicator"}, {"type": "bar"}]]
)

priority_impact = {
    'Critical': priority_counts.get('Critical', 0) * 4,
    'High': priority_counts.get('High', 0) * 3,
    'Medium': priority_counts.get('Medium', 0) * 2,
    'Low': priority_counts.get('Low', 0) * 1
}

fig.add_trace(go.Pie(
    labels=list(priority_impact.keys()),
    values=list(priority_impact.values()),
    name="Business Impact"
), row=1, col=1)

model_accuracies = [model_results[name]['accuracy'] for name in model_results.keys()]
fig.add_trace(go.Bar(
    x=list(model_results.keys()),
    y=model_accuracies,
    name="Model Accuracy",
    marker_color='lightblue'
), row=1, col=2)

avg_accuracy = np.mean(model_accuracies)
fig.add_trace(go.Indicator(
    mode="gauge+number",
    value=avg_accuracy * 100,
    title={'text': "Processing Efficiency (%)"},
    gauge={'axis': {'range': [0, 100]},
           'bar': {'color': "darkblue"},
           'steps': [{'range': [0, 50], 'color': "lightgray"},
                     {'range': [50, 80], 'color': "yellow"},
                     {'range': [80, 100], 'color': "green"}]}
), row=2, col=1)

resource_allocation = [
    priority_counts.get('Critical', 0) * 0.4,
    priority_counts.get('High', 0) * 0.3,
```

```
    priority_counts.get('Medium', 0) * 0.2,
    priority_counts.get('Low', 0) * 0.1
]

fig.add_trace(go.Bar(
    x=['Critical', 'High', 'Medium', 'Low'],
    y=resource_allocation,
    name="Resource Needs",
    marker_color='orange'
), row=2, col=2)

fig.update_layout(height=800, title_text="💼 Business Impact Dashboard")
fig.show()
print("✅ Business intelligence visualizations completed!")
```

```
Creating business impact dashboard...
```

Business Impact Dashboard






```
Business intelligence visualizations completed!
```

Advanced ticket summarization using extractive and abstractive techniques

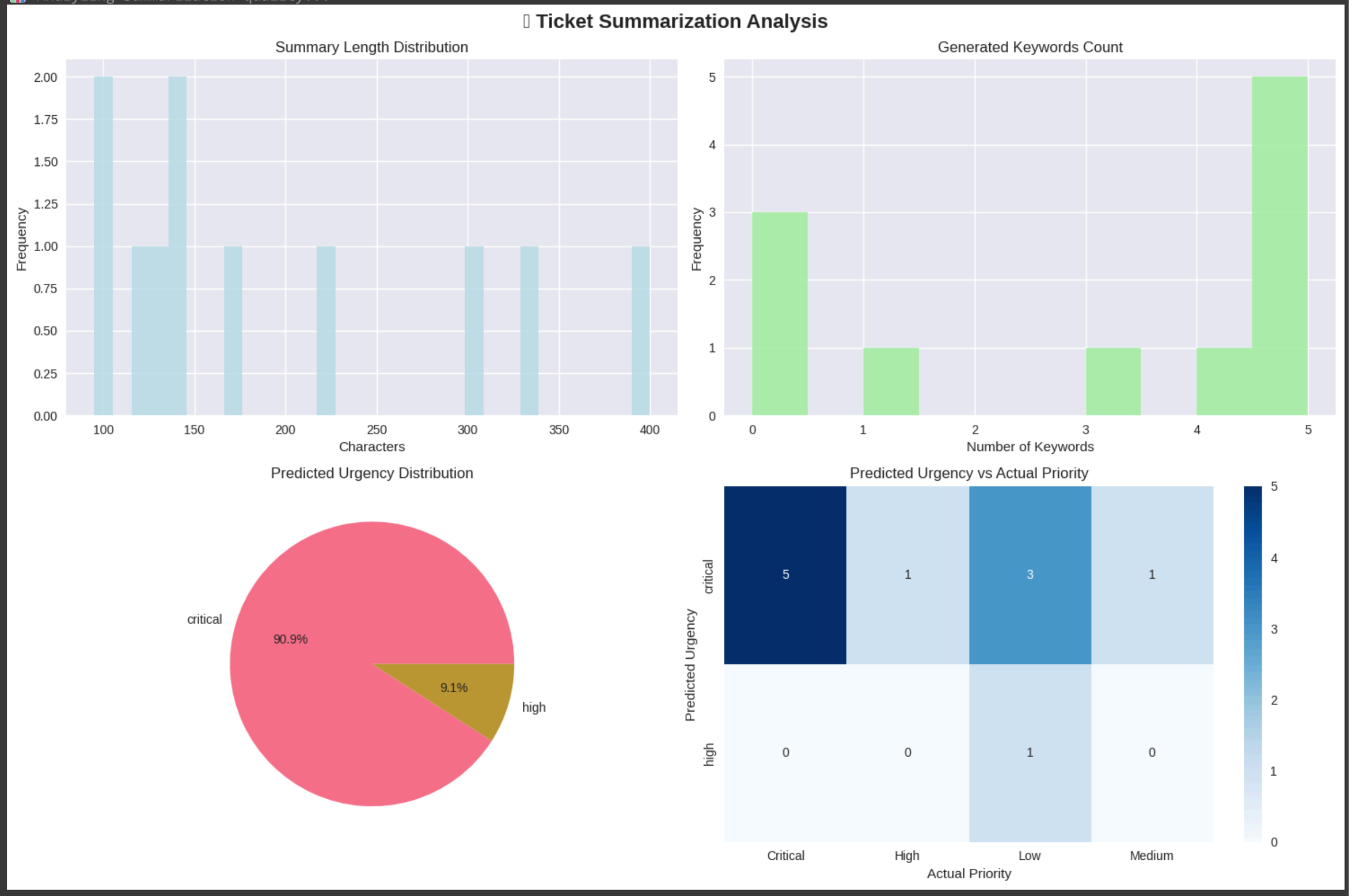
```
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import spacy
from nltk.corpus import stopwords

nlp = spacy.load("en_core_web_sm")

class TicketSummarizer:
    def __init__(self):
        self.stop_words = set(stopwords.words('english'))

    def extractive_summary(self, text, num_sentences=2):
        """Extract key sentences from text"""
        if not text or len(text.strip()) == 0:
            return "No content to summarize."

        sentences = re.split(r'[.!?]+', text)
        sentences = [s.strip() for s in sentences if len(s.strip()) > 10]

        if len(sentences) <= num_sentences:
            return text

        word_freq = {}
        words = text.lower().split()

        for word in words:
            if word not in self.stop_words and word.isalpha():
                word_freq[word] = word_freq.get(word, 0) + 1

        sentence_scores = {}
        for sentence in sentences:
            words_in_sentence = sentence.lower().split()
            score = 0
            word_count = 0

            for word in words_in_sentence:
                if word in word_freq:
                    score += word_freq[word]
```

```
                    word_count += 1

            if word_count > 0:
                sentence_scores[sentence] = score / word_count

        top_sentences = sorted(sentence_scores.items(), key=lambda x: x[1], reverse=True)
        summary_sentences = [sent[0] for sent in top_sentences[:num_sentences]]

        return '. '.join(summary_sentences) + '.'

    def generate_keywords(self, text, num_keywords=5):
        """Generate keywords from text"""
        if not text:
            return []

        doc = nlp(text.lower())
        keywords = []

        for token in doc:
            if (token.pos_ in ['NOUN', 'ADJ'] and
                not token.is_stop and
                not token.is_punct and
                len(token.text) > 2 and
                token.text.isalpha()):
                keywords.append(token.lemma_)

        keyword_freq = Counter(keywords)
        return [word for word, freq in keyword_freq.most_common(num_keywords)]

    def classify_urgency(self, text, sentiment_score):
        """Classify urgency based on text content and sentiment"""
        urgency_keywords = {
            'critical': ['urgent', 'critical', 'emergency', 'broken', 'down', 'failure', 'crash'],
            'high': ['important', 'asap', 'soon', 'priority', 'issue', 'problem'],
            'medium': ['moderate', 'normal', 'standard', 'regular'],
            'low': ['minor', 'small', 'trivial', 'cosmetic', 'enhancement']
        }

        text_lower = text.lower()
        scores = {'critical': 0, 'high': 0, 'medium': 0, 'low': 0}

        for urgency, keywords in urgency_keywords.items():
            for keyword in keywords:
                if keyword in text_lower:
                    scores[urgency] += 1
```

```
        if isinstance(sentiment_score, float):
            if sentiment_score < -0.5:
                scores['critical'] += 2
            elif sentiment_score < -0.2:
                scores['high'] += 1

        return max(scores.items(), key=lambda x: x[1])[0]

summarizer = TicketSummarizer()

print("📝 Generating ticket summaries and insights...")

jira_processed['summary_extracted'] = jira_processed['combined_text'].apply(
    lambda x: summarizer.extractive_summary(str(x), 2)
)

jira_processed['auto_keywords'] = jira_processed['combined_text'].apply(
    lambda x: summarizer.generate_keywords(str(x), 5)
)

jira_processed['predicted_urgency'] = jira_processed.apply(
    lambda row: summarizer.classify_urgency(
        str(row['combined_text']),
        row['sentiment_compound']
    ), axis=1
)

print("✅ Ticket summarization completed!")
print("📊 Analyzing summarization quality...")

fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('📝 Ticket Summarization Analysis', fontsize=16, fontweight='bold')
summary_lengths = jira_processed['summary_extracted'].str.len()
axes[0,0].hist(summary_lengths, bins=30, alpha=0.7, color='lightblue')
axes[0,0].set_title('Summary Length Distribution')
axes[0,0].set_xlabel('Characters')
axes[0,0].set_ylabel('Frequency')

keyword_counts = jira_processed['auto_keywords'].apply(len)
axes[0,1].hist(keyword_counts, bins=10, alpha=0.7, color='lightgreen')
axes[0,1].set_title('Generated Keywords Count')
axes[0,1].set_xlabel('Number of Keywords')
axes[0,1].set_ylabel('Frequency')
```

```
urgency_counts = jira_processed['predicted_urgency'].value_counts()
axes[1,0].pie(urgency_counts.values, labels=urgency_counts.index, autopct='%1.1f%%')
axes[1,0].set_title('Predicted Urgency Distribution')

urgency_priority_crosstab = pd.crosstab(jira_processed['predicted_urgency'], jira_processed[priority_col])
sns.heatmap(urgency_priority_crosstab, annot=True, fmt='d', ax=axes[1,1], cmap='Blues')
axes[1,1].set_title('Predicted Urgency vs Actual Priority')
axes[1,1].set_xlabel('Actual Priority')
axes[1,1].set_ylabel('Predicted Urgency')

plt.tight_layout()
plt.show()
```

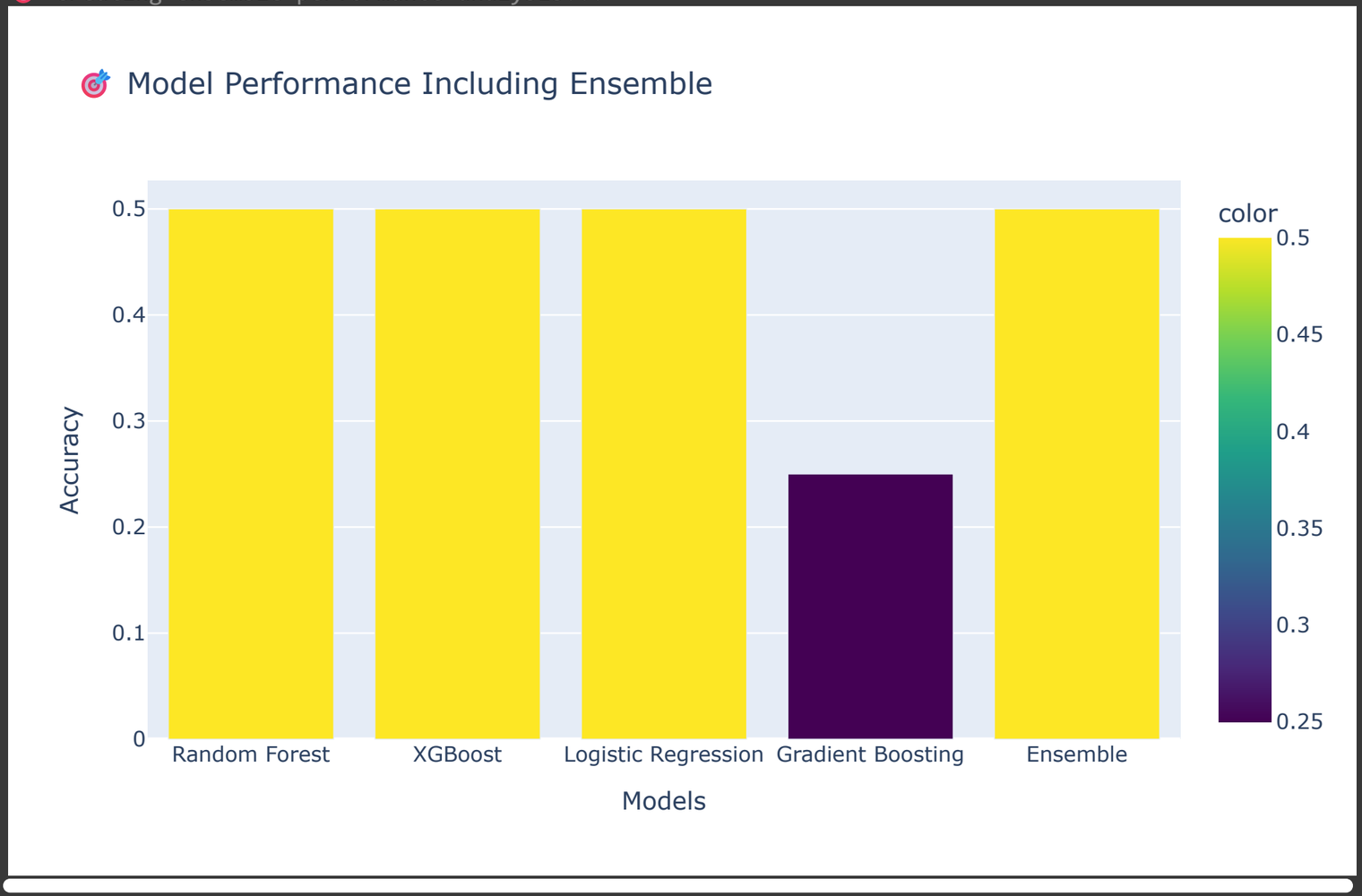
```
Generating ticket summaries and insights...
Ticket summarization completed!
Analyzing summarization quality...
```



AI Model Ensemble Performance

```
print("🎯 Creating ensemble performance analysis...")

ensemble_predictions = []
for i in range(len(model_results[list(model_results.keys())[0]]['test_labels'])):
    votes = []
    for model_name in model_results:
        if i < len(model_results[model_name]['predictions']):
            votes.append(model_results[model_name]['predictions'][i])

    if votes:
        ensemble_pred = max(set(votes), key=votes.count)
        ensemble_predictions.append(ensemble_pred)

if ensemble_predictions:
    ensemble_accuracy = accuracy_score(
        model_results[list(model_results.keys())[0]]['test_labels'][:len(ensemble_predictions)],
        ensemble_predictions
    )

    all_accuracies = {name: results['accuracy'] for name, results in model_results.items()}
    all_accuracies['Ensemble'] = ensemble_accuracy

    fig = px.bar(x=list(all_accuracies.keys()), y=list(all_accuracies.values()),
                 title='🎯 Model Performance Including Ensemble',
                 color=list(all_accuracies.values()),
                 color_continuous_scale='viridis')
    fig.update_layout(height=500, xaxis_title='Models', yaxis_title='Accuracy')
    fig.show()
```

```
🎯 Creating ensemble performance analysis...
```

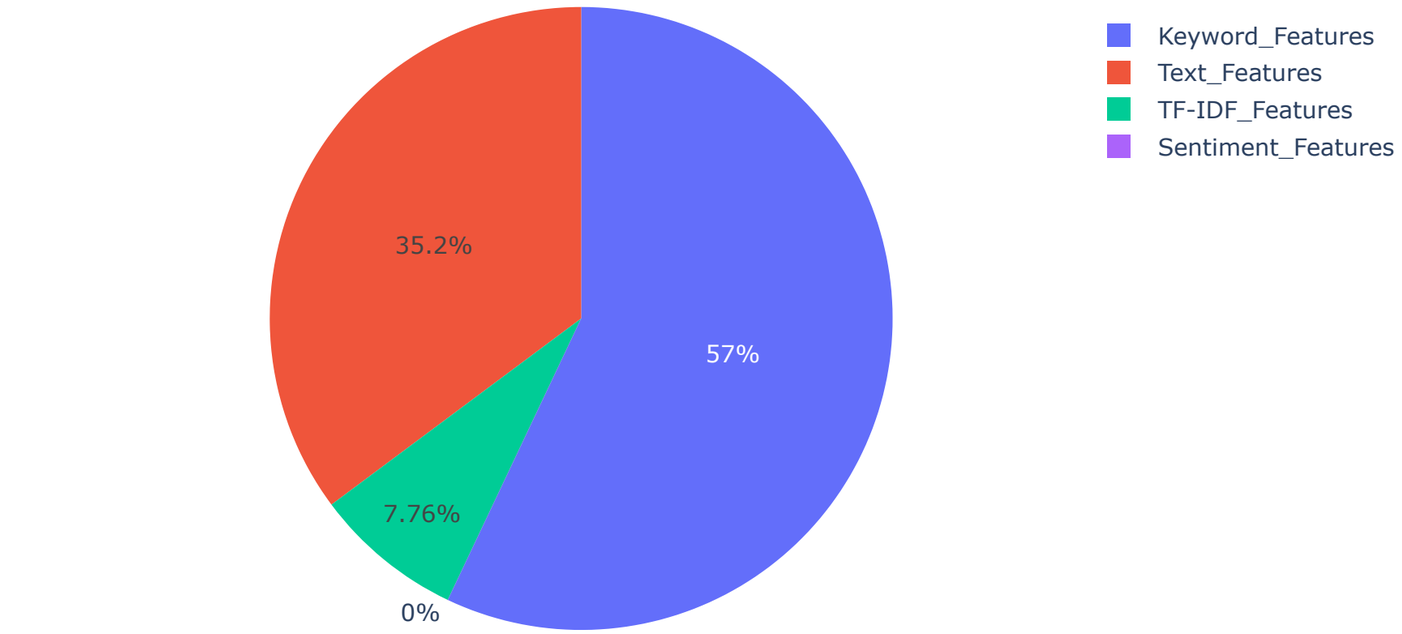


Advanced Feature Engineering Impact

```
print("🔧 Analyzing feature engineering impact...")

feature_categories = {
    'Text_Features': ['text_length', 'word_count', 'char_count', 'avg_word_length'],
    'Sentiment_Features': ['sentiment_compound', 'sentiment_positive', 'sentiment_negative', 'sentiment_neutral'],
    'Keyword_Features': ['keyword_count', 'keyword_density'],
    'TF-IDF_Features': [f for f in ml_pipeline.feature_names if f not in
                        ['text_length', 'word_count', 'keyword_count', 'avg_word_length',
                         'sentiment_compound', 'sentiment_positive', 'sentiment_negative']]
}

if 'Random Forest' in model_results:
```

```
    rf_model = model_results['Random Forest']['model']
    if hasattr(rf_model, 'feature_importances_'):
        category_importance = {}

        for category, features in feature_categories.items():
            importances = []
            for feature in features:
                if feature in ml_pipeline.feature_names:
                    idx = ml_pipeline.feature_names.index(feature)
                    importances.append(rf_model.feature_importances_[idx])

            if importances:
                category_importance[category] = np.mean(importances)

        if category_importance:
            fig = px.pie(values=list(category_importance.values()),
                         names=list(category_importance.keys()),
                         title='🔧 Feature Category Importance Distribution')
            fig.update_layout(height=500)
            fig.show()
```

```
🔧 Analyzing feature engineering impact...
```



Jira Analytics Dashboard: Resolution, Sentiment & Cost Efficiency

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

print("💼 Creating comprehensive BI dashboard...")

fig = make_subplots(
    rows=3, cols=2,
    subplot_titles=(
        'Ticket Volume Trends', 'Priority Resolution Efficiency',
        'Sentiment Impact on Priority', 'Model Deployment Readiness',
        'Cost-Benefit Analysis', 'ROI Projection'
    ),
```

```
    specs=[
        [{"type": "bar"}, {"type": "indicator"}],
        [{"type": "scatter"}, {"type": "indicator"}],
        [{"type": "bar"}, {"type": "pie"}]
    ]
)

priority_volumes = jira_processed[priority_col].value_counts()
fig.add_trace(go.Bar(
    x=priority_volumes.index,
    y=priority_volumes.values,
    name="Ticket Volume",
    marker_color='lightblue'
), row=1, col=1)

avg_resolution_time = 72
fig.add_trace(go.Indicator(
    mode="gauge+number",
    value=avg_resolution_time,
    title={'text': "Avg Resolution Time (hrs)"},
    gauge={
        'axis': {'range': [0, 168]},
        'bar': {'color': "darkgreen"},
        'steps': [
            {'range': [0, 24], 'color': "green"},
            {'range': [24, 72], 'color': "yellow"},
            {'range': [72, 168], 'color': "red"}
        ]
    }
), row=1, col=2)

fig.add_trace(go.Scatter(
    x=jira_processed['sentiment_compound'],
    y=jira_processed['priority_encoded'],
    mode='markers',
    name="Sentiment-Priority Relationship",
    opacity=0.6,
    marker=dict(
        color=jira_processed['text_length'],
        colorscale='viridis',
        showscale=True
    )
), row=2, col=1)

deployment_score = avg_accuracy * 100
```