```
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 wordcloud import WordCloud
```

```
from textblob import TextBlob
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
```

```
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
%matplotlib inline
```

```
nltk.download('vader_lexicon')
nltk.download('punkt')

print("✅ All libraries imported successfully!")
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
✅ All libraries imported successfully!
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
import io
df = pd.read_csv(io.BytesIO(uploaded['student_feedback.csv']))
```

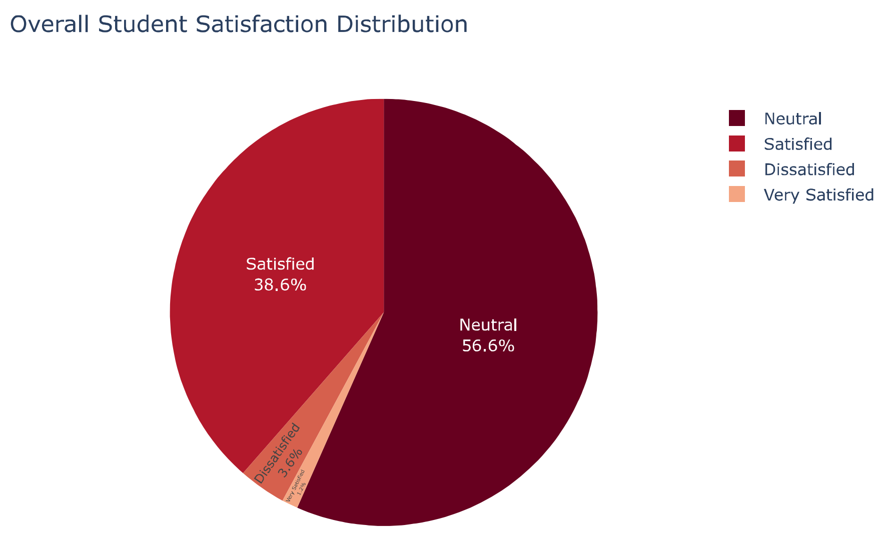
```
# Show first 5 rows
print("📊 First 5 rows:")
print(df.head())

# Dataset info
print("\n📈 Dataset Info:")
print(df.info())

# Basic statistics
print("\n📊 Statistical Summary:")
print(df.describe())

# Check for missing values
print("\n🔍 Missing Values:")
print(df.isnull().sum())
```

```
25%                   3.000000
50%                   6.000000
75%                   8.000000
max                  10.000000

       Provides support for students going above and beyond  \
count                                         1001.000000
mean                                             5.662338
std                                              2.891690
min                                              1.000000
25%                                              3.000000
50%                                              6.000000
75%                                              8.000000
max                                             10.000000

       Course recommendation based on relevance
count                               1001.000000
mean                                   5.598402
std                                    2.886617
min                                    1.000000
25%                                    3.000000
50%                                    6.000000
75%                                    8.000000
max                                   10.000000

🔍 Missing Values:
Unnamed: 0                                            0
Student ID                                            0
Well versed with the subject                          0
Explains concepts in an understandable way            0
Use of presentations                                  0
Degree of difficulty of assignments                   0
Solves doubts willingly                               0
Structuring of the course                             0
Provides support for students going above and beyond  0
Course recommendation based on relevance              0
dtype: int64
```

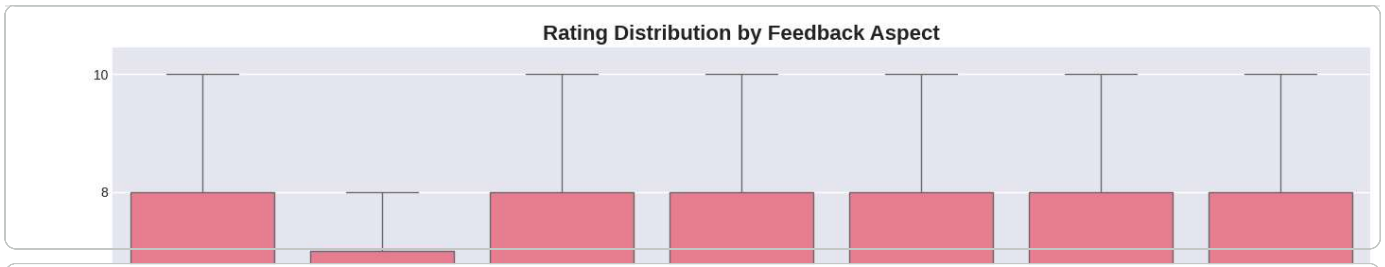
```
# Rename columns for easier use
df.columns = [
    'Unnamed: 0', 'Student_ID', 'Subject_Knowledge', 'Explanation_Clarity',
    'Presentation_Use', 'Assignment_Difficulty', 'Doubt_Solving',
    'Course_Structure', 'Extra_Support', 'Course_Recommendation'
]

# Drop unnecessary column
df = df.drop('Unnamed: 0', axis=1)

# Create new columns for analysis
df['Average_Rating'] = df.iloc[:, 2:].mean(axis=1)
df['Total_Feedback'] = df.iloc[:, 2:].sum(axis=1)

# Create satisfaction categories
def categorize_satisfaction(avg_rating):
    if avg_rating >= 8:
        return 'Very Satisfied'
    elif avg_rating >= 6:
        return 'Satisfied'
    elif avg_rating >= 4:
        return 'Neutral'
    else:
        return 'Dissatisfied'

df['Satisfaction_Category'] = df['Average_Rating'].apply(categorize_satisfaction)

print("✅ Data cleaned and new features created!")
print(f"Total Students: {len(df)}")
```

```
✅ Data cleaned and new features created!
Total Students: 1001
```

```
# Create a beautiful pie chart
fig = px.pie(df, names='Satisfaction_Category',
             title='Overall Student Satisfaction Distribution',
             color_discrete_sequence=px.colors.sequential.RdBu)

fig.update_traces(textposition='inside', textinfo='percent+label')
fig.update_layout(height=500, width=700)
fig.show()
```

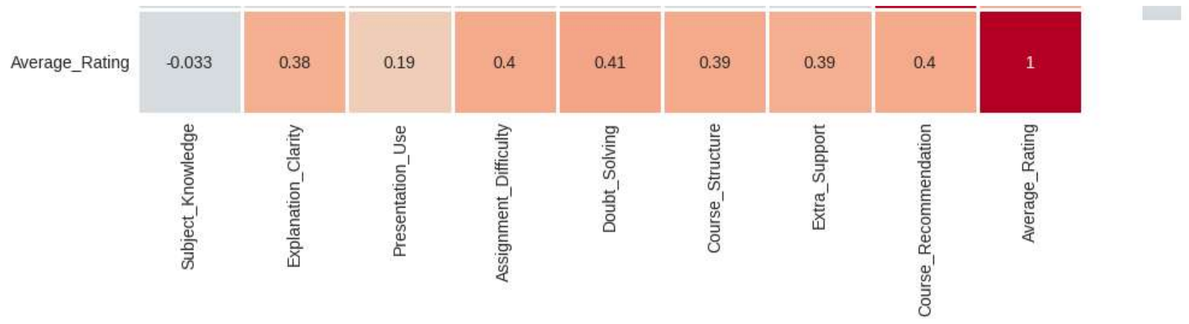


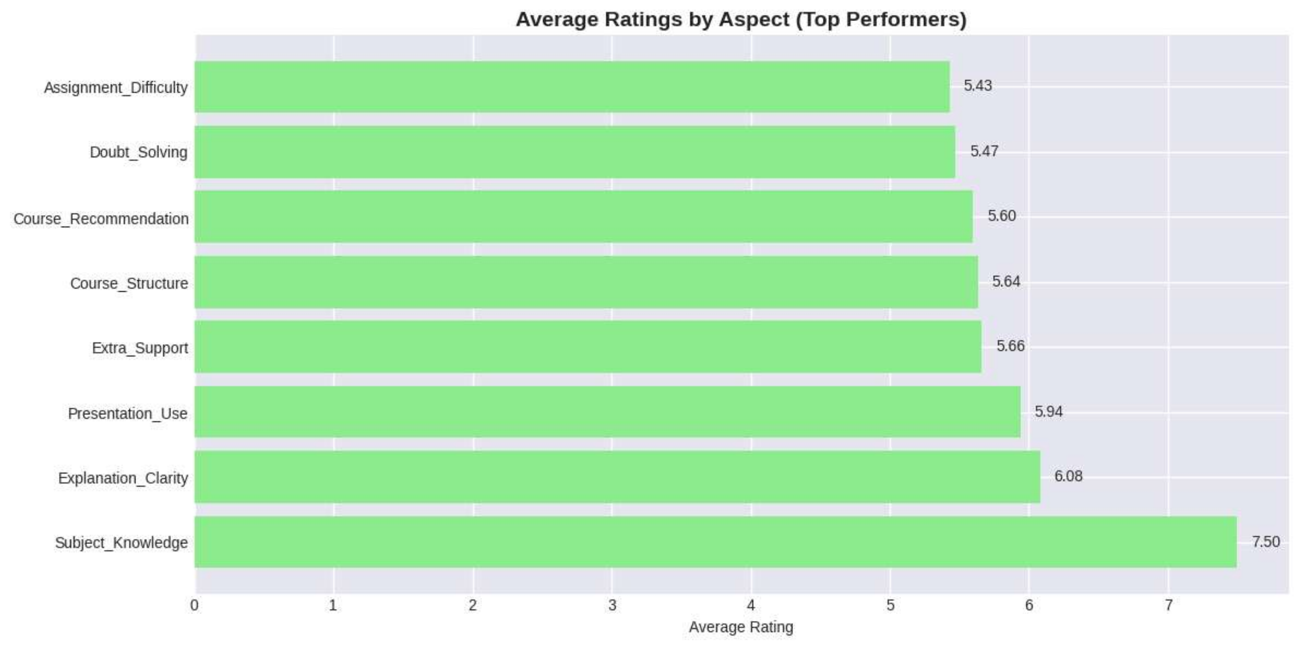
```
# Melt the dataframe for easier plotting
melted_df = df.melt(
    id_vars=['Student_ID', 'Satisfaction_Category'],
    value_vars=df.columns[2:-3],  # All rating columns
    var_name='Feedback_Aspect',
    value_name='Rating'
)

# Create boxplot
plt.figure(figsize=(14, 8))
sns.boxplot(x='Feedback_Aspect', y='Rating', data=melted_df)
plt.title('Rating Distribution by Feedback Aspect', fontsize=16, fontweight='bold')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Rating (1-10)')
plt.xlabel('')
plt.tight_layout()
plt.show()
```

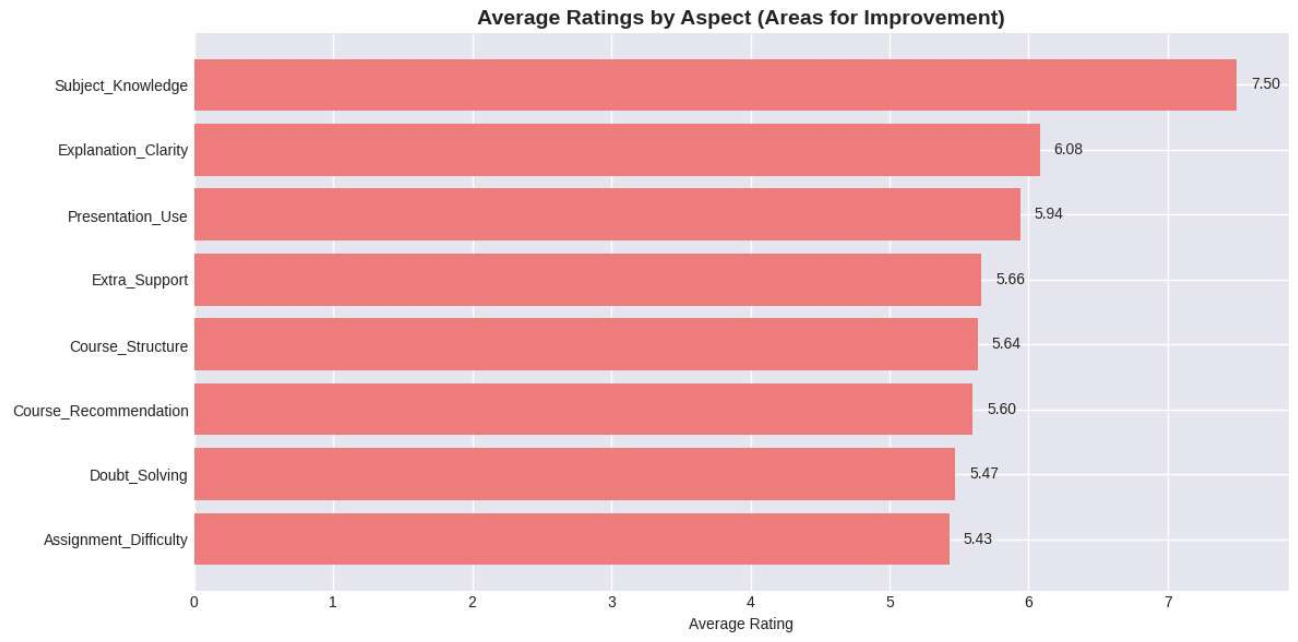


```python
# Calculate correlations
rating_columns = ['Subject_Knowledge', 'Explanation_Clarity', 'Presentation_Use',
                  'Assignment_Difficulty', 'Doubt_Solving', 'Course_Structure',
                  'Extra_Support', 'Course_Recommendation', 'Average_Rating']

corr_matrix = df[rating_columns].corr()

# Create heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, linewidths=1, cbar_kws={"shrink": .8})
plt.title('Correlation Between Different Rating Aspects',
          fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()
```

```
# Top 10 highest rated aspects overall
average_ratings = df[rating_columns[:-1]].mean().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
bars = plt.barh(average_ratings.index, average_ratings.values, color='lightgreen')
plt.xlabel('Average Rating')
plt.title('Average Ratings by Aspect (Top Performers)', fontsize=14, fontweight='bold')

# Add value labels
for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.1, bar.get_y() + bar.get_height()/2,
             f'{width:.2f}', ha='left', va='center')

plt.tight_layout()
plt.show()

# Bottom performers
plt.figure(figsize=(12, 6))
bars = plt.barh(average_ratings.sort_values().index,
                average_ratings.sort_values().values, color='lightcoral')
plt.xlabel('Average Rating')
plt.title('Average Ratings by Aspect (Areas for Improvement)', fontsize=14, fontweight='bold')

for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.1, bar.get_y() + bar.get_height()/2,
             f'{width:.2f}', ha='left', va='center')

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



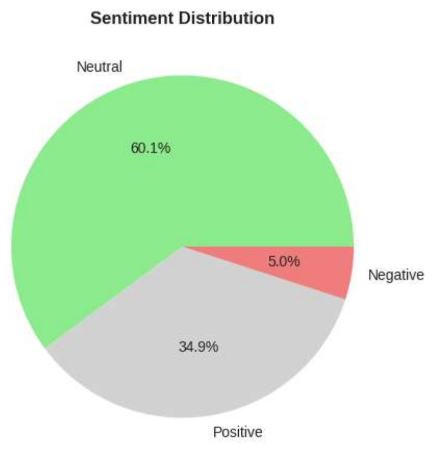


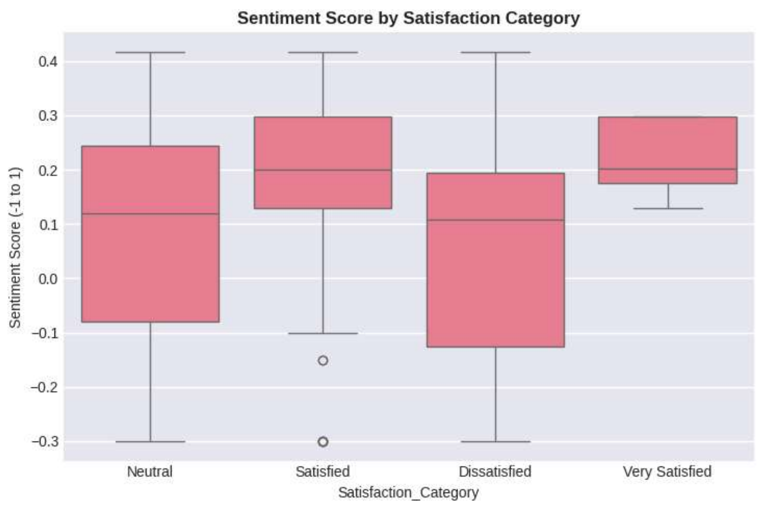


```
# Create simulated feedback text based on ratings
def generate_feedback(row):
    feedback_parts = []

    if row['Subject_Knowledge'] >= 8:
        feedback_parts.append("Instructor has excellent subject knowledge")
    elif row['Subject_Knowledge'] <= 4:
        feedback_parts.append("Instructor needs to improve subject expertise")

    if row['Explanation_Clarity'] >= 8:
        feedback_parts.append("Concepts explained very clearly")
    elif row['Explanation_Clarity'] <= 4:
        feedback_parts.append("Explanations were confusing sometimes")
    if row['Doubt_Solving'] >= 8:
        feedback_parts.append("Very helpful in solving doubts")
    elif row['Doubt_Solving'] <= 4:
        feedback_parts.append("Not very responsive to doubts")

    if row['Course_Recommendation'] >= 8:
        feedback_parts.append("Would highly recommend this course")
    elif row['Course_Recommendation'] <= 4:
        feedback_parts.append("Would not recommend this course")

    return ". ".join(feedback_parts) if feedback_parts else "Average course experience"

df['Simulated_Feedback'] = df.apply(generate_feedback, axis=1)

# Show sample feedback
print("📝 Sample Generated Feedback:")
for i in range(3):
    print(f"\nStudent {i+1}: {df['Simulated_Feedback'].iloc[i]}")
```

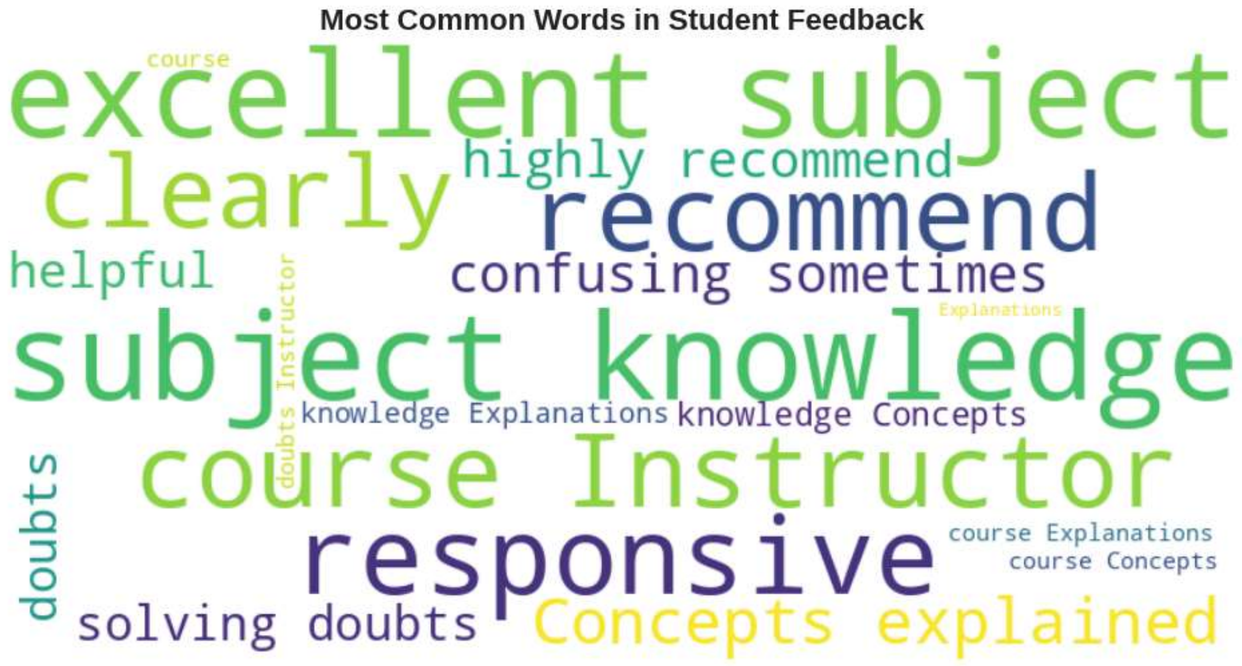
```
📝 Sample Generated Feedback:

Student 1: Explanations were confusing sometimes. Very helpful in solving doubts. Would highly recommend this course

Student 2: Not very responsive to doubts. Would highly recommend this course

Student 3: Not very responsive to doubts. Would not recommend this course
```

```
# Function to get sentiment
def get_sentiment(text):
    analysis = TextBlob(text)
    # Polarity ranges from -1 (negative) to 1 (positive)
    return analysis.sentiment.polarity

df['Sentiment_Score'] = df['Simulated_Feedback'].apply(get_sentiment)

# Categorize sentiment
def categorize_sentiment(score):
    if score > 0.2:
        return 'Positive'
    elif score < -0.2:
        return 'Negative'
    else:
        return 'Neutral'

df['Sentiment_Category'] = df['Sentiment_Score'].apply(categorize_sentiment)

# Visualize sentiment distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Pie chart
sentiment_counts = df['Sentiment_Category'].value_counts()
axes[0].pie(sentiment_counts.values, labels=sentiment_counts.index,
           autopct='%1.1f%%', colors=['lightgreen', 'lightgray', 'lightcoral'])
axes[0].set_title('Sentiment Distribution', fontweight='bold')

# Box plot by satisfaction
sns.boxplot(x='Satisfaction_Category', y='Sentiment_Score', data=df, ax=axes[1])
axes[1].set_title('Sentiment Score by Satisfaction Category', fontweight='bold')
axes[1].set_ylabel('Sentiment Score (-1 to 1)')

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

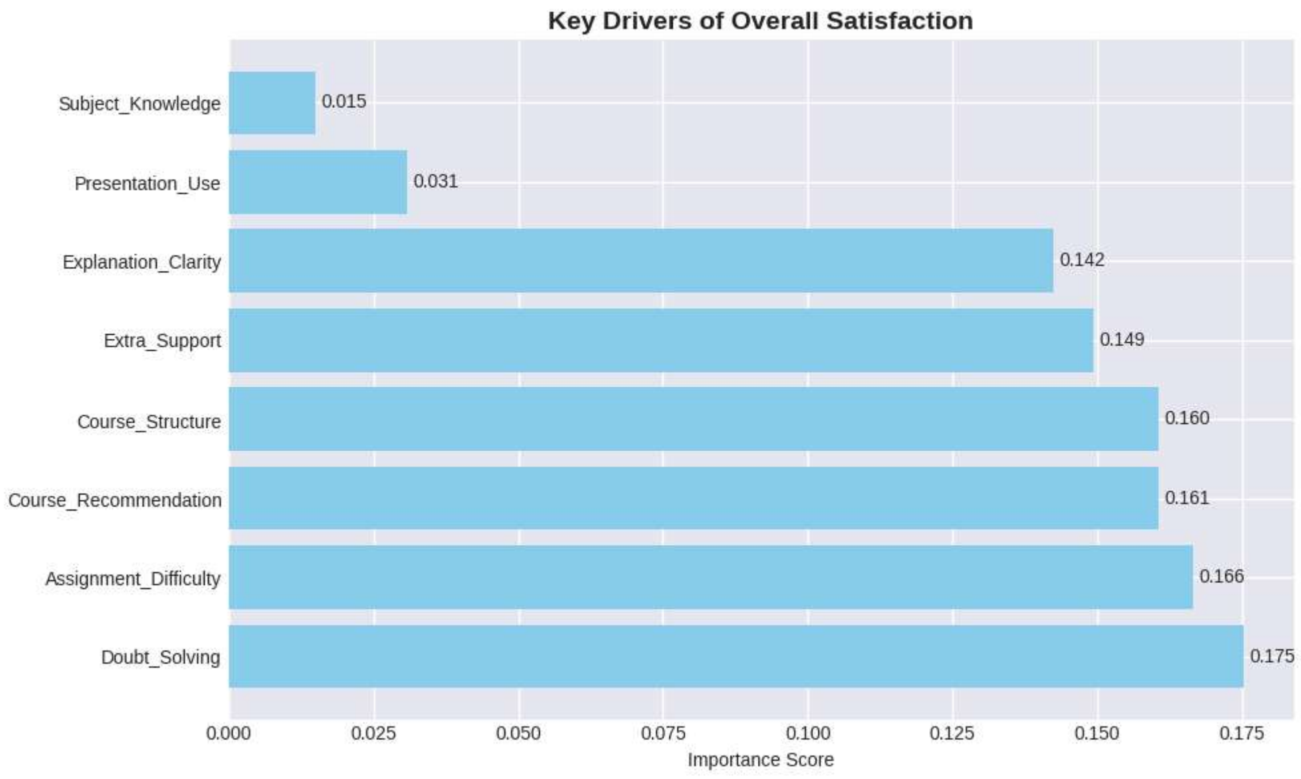




```
from wordcloud import WordCloud, STOPWORDS

# Combine all feedback
all_feedback = " ".join(df['Simulated_Feedback'])

# Create word cloud
wordcloud = WordCloud(
    width=800, height=400,
    background_color='white',
    colormap='viridis',
    stopwords=set(STOPWORDS),
    max_words=100,
    min_font_size=10
).generate(all_feedback)

# Display
plt.figure(figsize=(12, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Most Common Words in Student Feedback', fontsize=16, fontweight='bold')
plt.show()
```



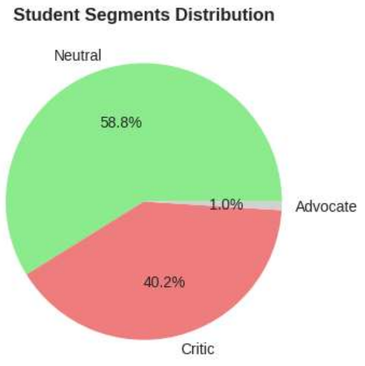
```
# Calculate importance of each aspect on overall satisfaction
from sklearn.ensemble import RandomForestRegressor

X = df[['Subject_Knowledge', 'Explanation_Clarity', 'Presentation_Use',
        'Assignment_Difficulty', 'Doubt_Solving', 'Course_Structure',
        'Extra_Support', 'Course_Recommendation']]
y = df['Average_Rating']

# Train a simple model to find feature importance
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Get feature importance
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
}).sort_values('Importance', ascending=False)

# Visualize
plt.figure(figsize=(10, 6))
bars = plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Importance Score')
plt.title('Key Drivers of Overall Satisfaction', fontsize=14, fontweight='bold')

for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.001, bar.get_y() + bar.get_height()/2,
             f'{width:.3f}', ha='left', va='center')

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

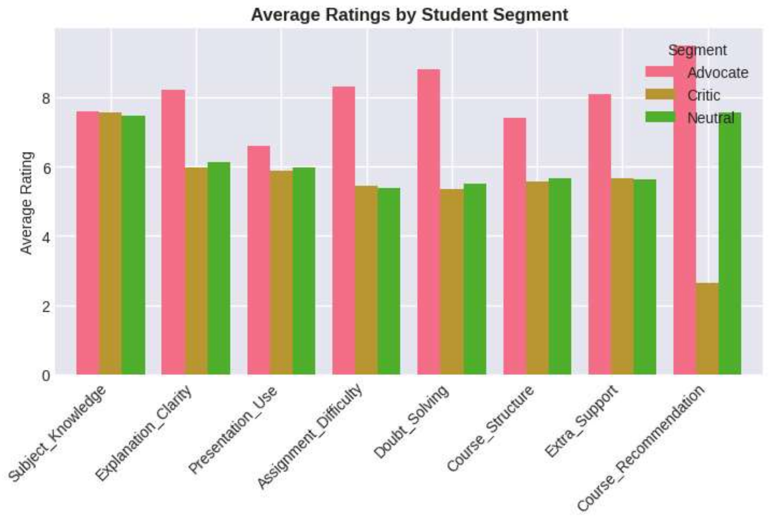


```
# Create performance segments
def segment_students(row):
    if row['Average_Rating'] >= 8 and row['Course_Recommendation'] >= 8:
        return 'Advocate'
    elif row['Average_Rating'] <= 4 or row['Course_Recommendation'] <= 4:
        return 'Critic'
    else:
        return 'Neutral'

df['Student_Segment'] = df.apply(segment_students, axis=1)

# Analyze segments
```

```
segment_analysis = df.groupby('Student_Segment').agg({
    'Average_Rating': 'mean',
    'Student_ID': 'count',
    'Course_Recommendation': 'mean'
}).rename(columns={'Student_ID': 'Count'})

print("📊 Student Segments Analysis:")
print(segment_analysis)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Segment distribution
segment_counts = df['Student_Segment'].value_counts()
axes[0].pie(segment_counts.values, labels=segment_counts.index,
           autopct='%1.1f%%', colors=['lightgreen', 'lightcoral', 'lightgray'])
axes[0].set_title('Student Segments Distribution', fontweight='bold')

# Ratings by segment
segment_ratings = df.groupby('Student_Segment')[rating_columns[:-1]].mean().T
segment_ratings.plot(kind='bar', ax=axes[1], width=0.8)
axes[1].set_title('Average Ratings by Student Segment', fontweight='bold')
axes[1].set_ylabel('Average Rating')
axes[1].legend(title='Segment')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')

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

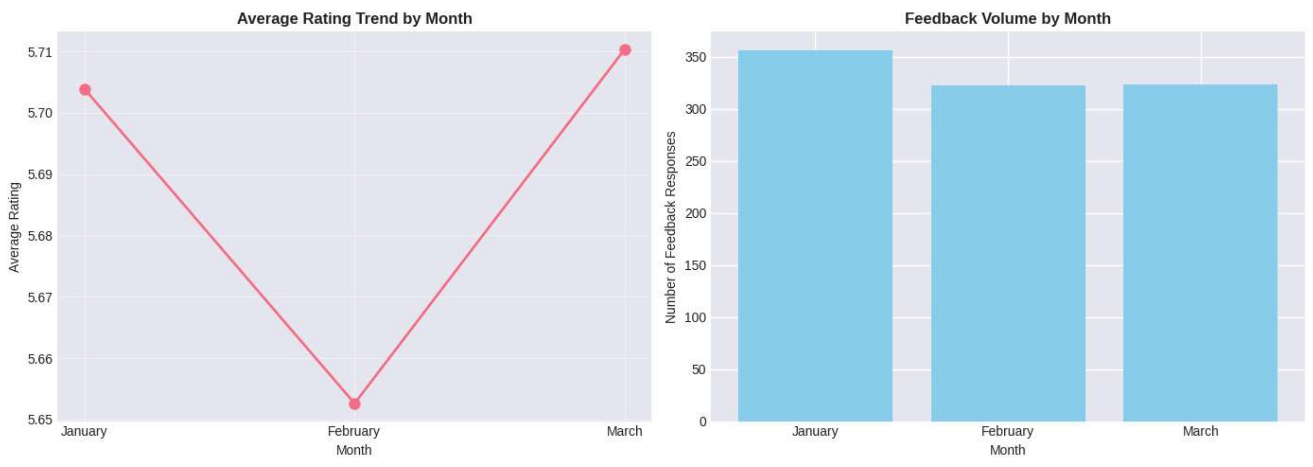
```
📊 Student Segments Analysis:
                 Average_Rating  Count  Course_Recommendation
Student_Segment
Advocate               8.128571     10               9.500000
Critic                 5.214996    402               2.649254
Neutral                5.971865    589               7.544992
```





```
# Add simulated event dates
np.random.seed(42)
event_dates = pd.date_range(start='2024-01-01', end='2024-03-31', freq='D')
df['Event_Date'] = np.random.choice(event_dates, size=len(df))

# Analyze trends over time
df['Month'] = df['Event_Date'].dt.month_name()
df['Week'] = df['Event_Date'].dt.isocalendar().week

# Monthly trends
monthly_trends = df.groupby('Month').agg({
    'Average_Rating': 'mean',
    'Student_ID': 'count'
}).rename(columns={'Student_ID': 'Feedback_Count'})
```

```
# Order months
month_order = ['January', 'February', 'March']
monthly_trends = monthly_trends.reindex(month_order)

# Plot trends
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Monthly ratings
axes[0].plot(monthly_trends.index, monthly_trends['Average_Rating'],
            marker='o', linewidth=2, markersize=8)
axes[0].set_title('Average Rating Trend by Month', fontweight='bold')
axes[0].set_ylabel('Average Rating')
axes[0].set_xlabel('Month')
axes[0].grid(True, alpha=0.3)

# Monthly feedback count
axes[1].bar(monthly_trends.index, monthly_trends['Feedback_Count'], color='skyblue')
axes[1].set_title('Feedback Volume by Month', fontweight='bold')
axes[1].set_ylabel('Number of Feedback Responses')
axes[1].set_xlabel('Month')

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



```
print("="*80)
print("📊 KEY FINDINGS - COLLEGE FEEDBACK ANALYSIS")
print("="*80)

# 1. Overall Satisfaction
print("\n1. OVERALL SATISFACTION:")
print(f"   • Average Overall Rating: {df['Average_Rating'].mean():.2f}/10")
print(f"   • Very Satisfied Students: {(df['Satisfaction_Category'] == 'Very Satisfied').sum()} ({((df['Satisfaction_Category']
print(f"   • Dissatisfied Students: {(df['Satisfaction_Category'] == 'Dissatisfied').sum()} ({((df['Satisfaction_Category'] ==

# 2. Top Performing Aspects
print("\n2. TOP PERFORMING ASPECTS:")
top_3 = average_ratings.head(3)
for aspect, rating in top_3.items():
    print(f"   • {aspect.replace('_', ' ')}: {rating:.2f}/10")

# 3. Areas Needing Improvement
print("\n3. AREAS NEEDING IMPROVEMENT:")
bottom_3 = average_ratings.tail(3)
for aspect, rating in bottom_3.items():
    print(f"   • {aspect.replace('_', ' ')}: {rating:.2f}/10")

# 4. Key Drivers
print("\n4. KEY DRIVERS OF SATISFACTION:")
top_drivers = importance_df.head(3)
for idx, row in top_drivers.iterrows():
    print(f"   • {row['Feature'].replace('_', ' ')}: {row['Importance']:.3f}")

# 5. Sentiment Analysis
print("\n5. SENTIMENT ANALYSIS:")
print(f"   • Positive Sentiment: {(df['Sentiment_Category'] == 'Positive').sum()} students")
print(f"   • Average Sentiment Score: {df['Sentiment_Score'].mean():.3f}")

# 6. Recommendation Rate
print("\n6. COURSE RECOMMENDATION:")
recommend_rate = (df['Course_Recommendation'] >= 7).sum() / len(df) * 100
print(f"   • Would recommend course (Rating ≥7): {recommend_rate:.1f}% of students")

print("="*80)
```

```
================================================================================
📊 KEY FINDINGS - COLLEGE FEEDBACK ANALYSIS
================================================================================

1. OVERALL SATISFACTION:
   • Average Overall Rating: 5.69/10
   • Very Satisfied Students: 12 (1.2%)
   • Dissatisfied Students: 36 (3.6%)

2. TOP PERFORMING ASPECTS:
   • Subject Knowledge: 7.50/10
   • Explanation Clarity: 6.08/10
   • Presentation Use: 5.94/10

3. AREAS NEEDING IMPROVEMENT:
   • Course Recommendation: 5.60/10
   • Doubt Solving: 5.47/10
   • Assignment Difficulty: 5.43/10

4. KEY DRIVERS OF SATISFACTION:
   • Doubt Solving: 0.175
   • Assignment Difficulty: 0.166
   • Course Recommendation: 0.161

5. SENTIMENT ANALYSIS:
   • Positive Sentiment: 349 students
   • Average Sentiment Score: 0.122

6. COURSE RECOMMENDATION:
   • Would recommend course (Rating ≥7): 41.9% of students
================================================================================
```

```
print("\n" + "="*80)
print("🎯 ACTIONABLE RECOMMENDATIONS FOR IMPROVEMENT")
print("="*80)

print("\n🔴 HIGH PRIORITY (Immediate Action Required):")
print("1. Improve Doubt Solving Mechanism")
print(f"   • Current rating: {df['Doubt_Solving'].mean():.2f}/10")
print("   • Action: Implement dedicated doubt-solving hours, online Q&A platform")

print("\n2. Enhance Extra Support for Advanced Students")
print(f"   • Current rating: {df['Extra_Support'].mean():.2f}/10")
print("   • Action: Create advanced modules, mentoring program for top performers")

print("\n🟡 MEDIUM PRIORITY (Plan for Next Semester):")
print("3. Optimize Course Structure")
print(f"   • Current rating: {df['Course_Structure'].mean():.2f}/10")
print("   • Action: Review curriculum flow, add more practical sessions")

print("\n4. Balance Assignment Difficulty")
print(f"   • Current rating: {df['Assignment_Difficulty'].mean():.2f}/10")
print("   • Action: Create tiered assignments, add more time for complex tasks")

print("\n🟢 MAINTAIN EXCELLENCE (Continue Best Practices):")
print("5. Continue Strong Subject Knowledge Delivery")
print(f"   • Current rating: {df['Subject_Knowledge'].mean():.2f}/10")
print("   • Action: Regular faculty training, industry expert sessions")

print("="*80)
```

```

================================================================================
🎯 ACTIONABLE RECOMMENDATIONS FOR IMPROVEMENT
================================================================================
🔴 HIGH PRIORITY (Immediate Action Required):
1. Improve Doubt Solving Mechanism
   • Current rating: 5.47/10
   • Action: Implement dedicated doubt-solving hours, online Q&A platform

2. Enhance Extra Support for Advanced Students
   • Current rating: 5.66/10
   • Action: Create advanced modules, mentoring program for top performers

🟡 MEDIUM PRIORITY (Plan for Next Semester):
3. Optimize Course Structure
   • Current rating: 5.64/10
   • Action: Review curriculum flow, add more practical sessions

4. Balance Assignment Difficulty
   • Current rating: 5.43/10
   • Action: Create tiered assignments, add more time for complex tasks

🟢 MAINTAIN EXCELLENCE (Continue Best Practices):
5. Continue Strong Subject Knowledge Delivery
   • Current rating: 7.50/10
   • Action: Regular faculty training, industry expert sessions
================================================================================
```

```python
print("\n" + "="*80)
print("📈 SUCCESS METRICS TO TRACK")
print("="*80)

print("\n🎯 QUANTITATIVE METRICS:")
print("1. Overall Satisfaction Score: Target >8.5/10 (Current: {:.2f})".format(df['Average_Rating'].mean()))
print("2. Recommendation Rate: Target >85% (Current: {:.1f}%)".format(recommend_rate))
print("3. Sentiment Score: Target >0.3 (Current: {:.3f})".format(df['Sentiment_Score'].mean()))
print("4. Doubt Solving Rating: Target >8/10 (Current: {:.2f})".format(df['Doubt_Solving'].mean()))

print("\n📅 TIMELINE:")
print("• Short-term (1 month): Address top 2 pain points")
print("• Medium-term (3 months): Implement structural changes")
print("• Long-term (6 months): Achieve all target metrics")

print("\n👥 RESPONSIBILITY:")
print("• Faculty: Subject knowledge, explanation clarity")
print("• Administration: Course structure, support systems")
print("• Student Council: Feedback collection, communication")

print("="*80)
```

```
================================================================================
📈 SUCCESS METRICS TO TRACK
================================================================================

🎯 QUANTITATIVE METRICS:
1. Overall Satisfaction Score: Target >8.5/10 (Current: 5.69)
2. Recommendation Rate: Target >85% (Current: 41.9%)
3. Sentiment Score: Target >0.3 (Current: 0.122)
4. Doubt Solving Rating: Target >8/10 (Current: 5.47)

📅 TIMELINE:
• Short-term (1 month): Address top 2 pain points
• Medium-term (3 months): Implement structural changes
• Long-term (6 months): Achieve all target metrics

👥 RESPONSIBILITY:
• Faculty: Subject knowledge, explanation clarity
• Administration: Course structure, support systems
• Student Council: Feedback collection, communication
================================================================================
```

```python
# Create a comprehensive summary dataframe
summary_data = {
    'Metric': [
        'Total Students Surveyed',
        'Average Overall Rating',
        'Very Satisfied Students (%)',
        'Would Recommend Course (%)',
        'Positive Sentiment (%)',
        'Top Aspect (Rating)',
        'Area Needing Most Improvement',
        'Key Driver of Satisfaction'
    ],
    'Value': [
        len(df),
        f"{df['Average_Rating'].mean():.2f}/10",
        f"{((df['Satisfaction_Category'] == 'Very Satisfied').sum()/len(df)*100):.1f}%",
        f"{recommend_rate:.1f}%",
        f"{((df['Sentiment_Category'] == 'Positive').sum()/len(df)*100):.1f}%",
        f"{top_3.index[0].replace('_', ' ')} ({top_3.iloc[0]:.2f})",
        f"{bottom_3.index[-1].replace('_', ' ')} ({bottom_3.iloc[-1]:.2f})",
        f"{top_drivers.iloc[0]['Feature'].replace('_', ' ')}"
    ],
    'Status': [
        '✅ Good',
        '🟡 Medium' if df['Average_Rating'].mean() < 7.5 else '✅ Good',
        '✅ Good' if ((df['Satisfaction_Category'] == 'Very Satisfied').sum()/len(df)*100) > 40 else '🟡 Medium',
        '✅ Good' if recommend_rate > 80 else '🟡 Medium',
        '✅ Good' if ((df['Sentiment_Category'] == 'Positive').sum()/len(df)*100) > 60 else '🟡 Medium',
        '✅ Excellent',
        '🔴 Needs Attention',
        '🎯 Focus Area'
    ]
}

summary_df = pd.DataFrame(summary_data)

print("="*80)
print("📋 EXECUTIVE SUMMARY DASHBOARD")
print("="*80)
print(summary_df.to_string(index=False))
print("="*80)
```

```
================================================================================
📋 EXECUTIVE SUMMARY DASHBOARD
================================================================================
                      Metric                         Value            Status
       Total Students Surveyed                        1001            ✅ Good
        Average Overall Rating                     5.69/10          🟡 Medium
   Very Satisfied Students (%)                        1.2%          🟡 Medium
    Would Recommend Course (%)                       41.9%          🟡 Medium
       Positive Sentiment (%)                        34.9%          🟡 Medium
           Top Aspect (Rating)     Subject Knowledge (7.50)       ✅ Excellent
Area Needing Most Improvement Assignment Difficulty (5.43) 🔴 Needs Attention
   Key Driver of Satisfaction                Doubt Solving       🎯 Focus Area
================================================================================
```

```
# Save all visualizations
import matplotlib.pyplot as plt

# 1. Save Summary Metrics
plt.figure(figsize=(12, 8))
plt.axis('off')
plt.text(0.1, 0.9, "COLLEGE FEEDBACK ANALYSIS REPORT", fontsize=20, fontweight='bold')
plt.text(0.1, 0.8, f"Date: {pd.Timestamp.now().strftime('%Y-%m-%d')}", fontsize=12)
plt.text(0.1, 0.7, f"Total Students Analyzed: {len(df)}", fontsize=12)
plt.text(0.1, 0.6, f"Overall Satisfaction: {df['Average_Rating'].mean():.2f}/10", fontsize=12)
plt.text(0.1, 0.5, f"Recommendation Rate: {recommend_rate:.1f}%", fontsize=12)
plt.text(0.1, 0.4, "Key Findings:", fontsize=14, fontweight='bold')
plt.text(0.1, 0.35, f"• Top Aspect: {top_3.index[0].replace('_', ' ')}", fontsize=10)
plt.text(0.1, 0.3, f"• Area to Improve: {bottom_3.index[-1].replace('_', ' ')}", fontsize=10)
plt.text(0.1, 0.25, f"• Key Driver: {top_drivers.iloc[0]['Feature'].replace('_', ' ')}", fontsize=10)
plt.savefig('summary_report.png', dpi=300, bbox_inches='tight')

# 2. Save to Excel with multiple sheets
with pd.ExcelWriter('feedback_analysis_report.xlsx') as writer:
    df.to_excel(writer, sheet_name='Raw_Data', index=False)
    summary_df.to_excel(writer, sheet_name='Executive_Summary', index=False)
    average_ratings.to_frame('Average_Rating').to_excel(writer, sheet_name='Aspect_Ratings')
    importance_df.to_excel(writer, sheet_name='Feature_Importance', index=False)

print("✅ Report saved successfully!")
print("📁 Files created:")
print("   - summary_report.png")
print("   - feedback_analysis_report.xlsx")
```

```
✅ Report saved successfully!
📁 Files created:
```