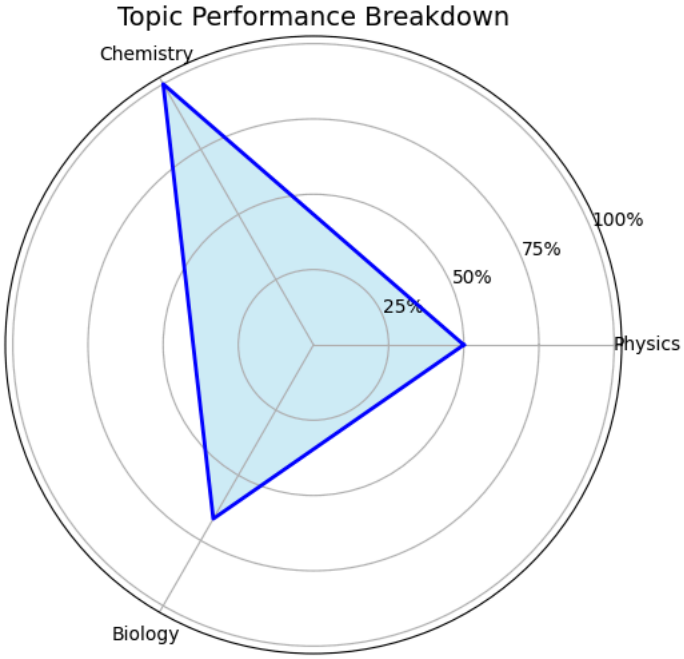
```
pip install pandas numpy matplotlib seaborn scikit-learn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.55.7)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

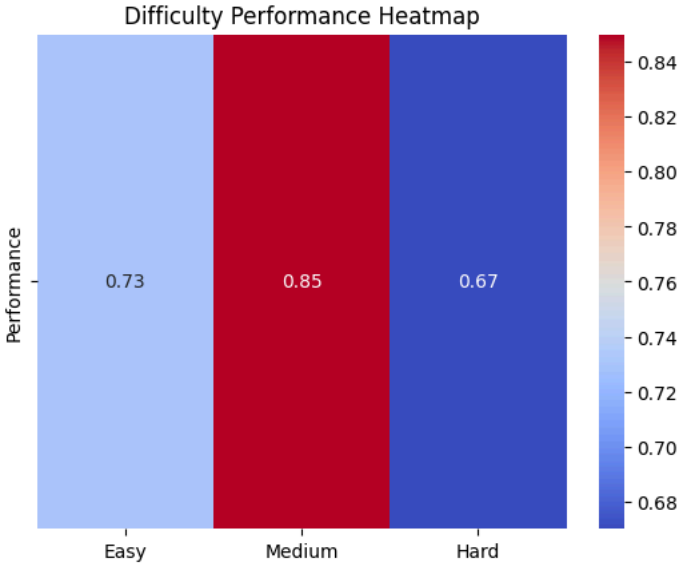
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

def advanced_output(insights, recommendations, rank_prediction):
    print("### Performance Insights ###")
    print(f"Overall Score: {insights['current']['overall_score']}%")

    # Topic Performance Breakdown
    topic_performance = insights['current']['topic_performance']
    print("\nTopic-Wise Performance:")
    for topic, score in topic_performance.items():
        print(f"  - {topic}: {score:.2f}%")

    # Time Management Insights
    time_mgmt = insights['current']['time_management']
    print("\nTime Management:")
    print(f"  - Average Time per Question: {time_mgmt['avg_time']} seconds")
    print(f"  - Maximum Time Taken: {time_mgmt['max_time']} seconds")
    print(f"  - Minimum Time Taken: {time_mgmt['min_time']} seconds")

    # Generate Spider Chart for Topic Performance
    topics = list(topic_performance.keys())
    scores = list(topic_performance.values())
    scores.append(scores[0])  # Closing the circle
    angles = np.linspace(0, 2 * np.pi, len(scores), endpoint=True)

    fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
    ax.fill(angles, scores, color='skyblue', alpha=0.4)
    ax.plot(angles, scores, color='blue', linewidth=2)
    ax.set_yticks([25, 50, 75, 100])
    ax.set_yticklabels(['25%', '50%', '75%', '100%'])
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(topics)
    plt.title("Topic Performance Breakdown", size=14)
    plt.show()

    # Improvement Trends
    print("\n### Historical Performance Trends ###")
    print("Performance Over Last 5 Quizzes:")
    print(f"  - Scores: {insights['historical']['trend']}")

    # Difficulty Level Trends
    difficulty_trends = insights['historical']['difficulty_performance']
    print("\nPerformance by Difficulty Level:")
    for level, performance in difficulty_trends.items():
        print(f"  - {level}: {performance * 100:.2f}%")

    # Heatmap for Topic Trends
    sns.heatmap(
        np.array([[difficulty_trends['Easy'], difficulty_trends['Medium'], difficulty_trends['Hard']]]),
        annot=True,
        xticklabels=['Easy', 'Medium', 'Hard'],
        yticklabels=['Performance'],
        cmap='coolwarm'
    )
    plt.title("Difficulty Performance Heatmap")
    plt.show()
```

```
        # Recommendations
        print("\n### Personalized Recommendations ###")
        for rec in recommendations:
            print(f"  - {rec}")

        # NEET Rank Prediction
        print("\n### NEET Rank Prediction ###")
        print(f"Predicted Rank: {rank_prediction['predicted_rank']}")
        print(f"Confidence Score: {rank_prediction['confidence_score']:.2f}")

        # Rank Prediction Confidence Graph
        ranks = list(range(1, 50000, 1000))
        confidence_scores = [np.exp(-0.0001 * (rank - rank_prediction['predicted_rank'])**2) for rank in ranks]
        plt.figure(figsize=(8, 4))
        plt.plot(ranks, confidence_scores, label="Confidence")
        plt.axvline(x=rank_prediction['predicted_rank'], color='red', linestyle='--', label="Predicted Rank")
        plt.title("NEET Rank Confidence Distribution")
        plt.xlabel("Rank")
        plt.ylabel("Confidence Score")
        plt.legend()
        plt.show()

    # Sample Input for Testing
    insights = {
        'current': {
            'overall_score': 70.0,
            'topic_performance': {'Physics': 50.0, 'Chemistry': 100.0, 'Biology': 66.67},
            'time_management': {'avg_time': 45.5, 'max_time': 65, 'min_time': 30}
        },
        'historical': {
            'trend': [90.0, 70.0, 40.0, 90.0, 90.0],
            'topic_trends': {'Chemistry': 85.0, 'Biology': 66.67, 'Physics': 73.33},
            'difficulty_performance': {'Medium': 0.85, 'Hard': 0.67, 'Easy': 0.73}
        }
    }

    recommendations = [
        "Focus on strengthening your understanding of Physics. Consider dedicating extra study time to this subject.",
        "Focus on strengthening your understanding of Biology. Consider dedicating extra study time to this subject."
    ]

    rank_prediction = {
        'predicted_rank': 33216,
        'confidence_score': 0.846
    }

    advanced_output(insights, recommendations, rank_prediction)
```

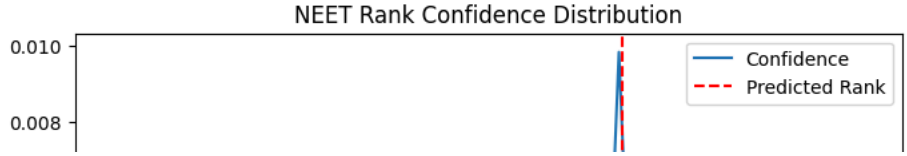
```
### Performance Insights ###
Overall Score: 70.0%

Topic-Wise Performance:
  - Physics: 50.00%
  - Chemistry: 100.00%
  - Biology: 66.67%

Time Management:
  - Average Time per Question: 45.5 seconds
  - Maximum Time Taken: 65 seconds
  - Minimum Time Taken: 30 seconds
```

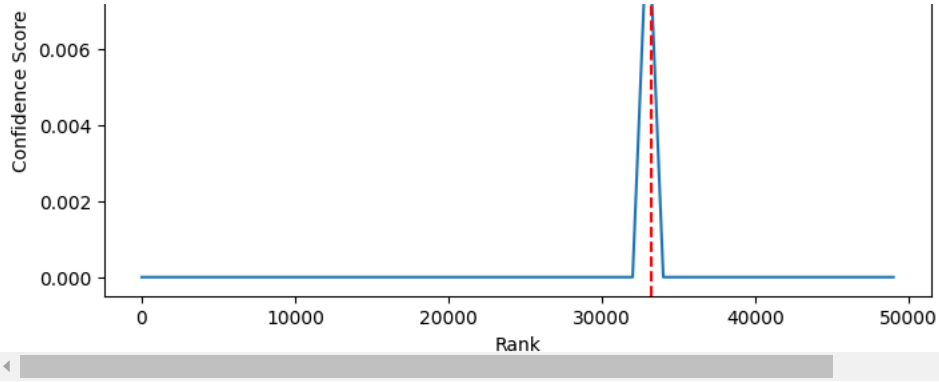



```
### Historical Performance Trends ###
Performance Over Last 5 Quizzes:
  - Scores: [90.0, 70.0, 40.0, 90.0, 90.0]

Performance by Difficulty Level:
  - Medium: 85.00%
  - Hard: 67.00%
  - Easy: 73.00%
```




```
### Personalized Recommendations ###
  - Focus on strengthening your understanding of Physics. Consider dedicating extra study time to this subject.
  - Focus on strengthening your understanding of Biology. Consider dedicating extra study time to this subject.

### NEET Rank Prediction ###
Predicted Rank: 33216
Confidence Score: 0.85
```





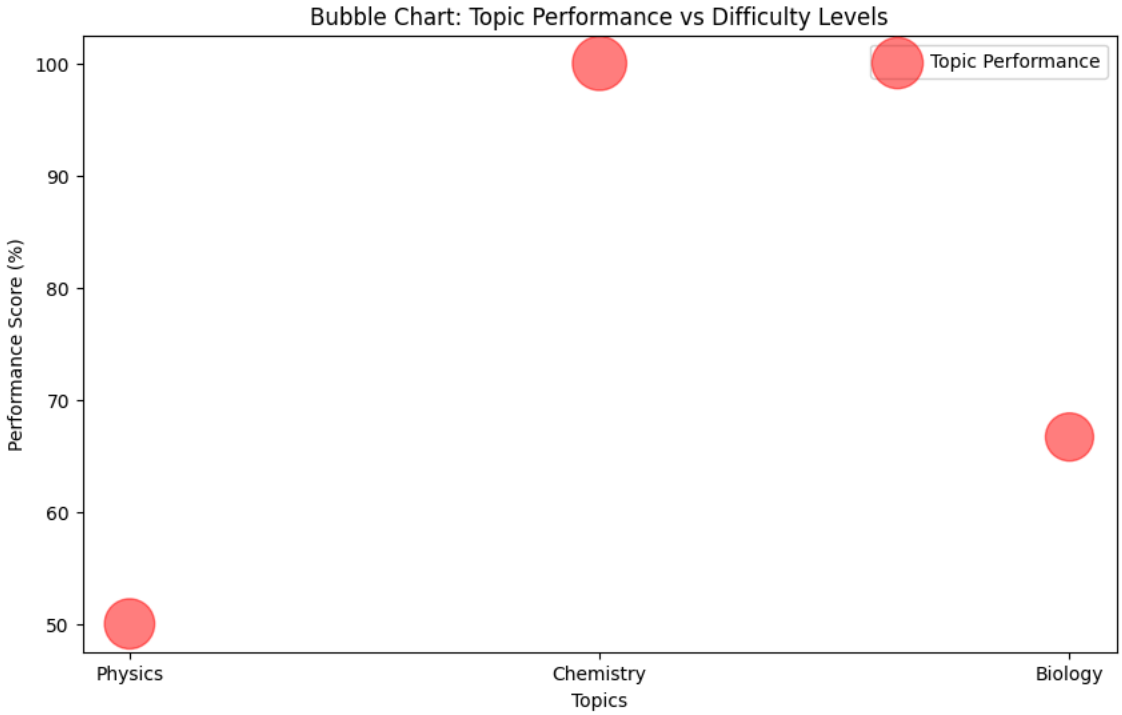

```
import matplotlib.pyplot as plt
import numpy as np

def generate_advanced_output(insights, recommendations):
    # Displaying Performance Insights
    print("### Performance Insights ###")
    print(f"Overall Score: {insights['current']['overall_score']}%")

    # Topic-wise Performance Breakdown
    print("\nTopic-Wise Performance:")
    for topic, score in insights['current']['topic_performance'].items():
        print(f"  - {topic}: {score}%")

    # Difficulty-wise Performance Breakdown
    print("\nDifficulty-Level Performance:")
    for difficulty, score in insights['current']['difficulty_performance'].items():
        print(f"  - {difficulty}: {score * 100:.2f}%")

    # Historical Performance Trends
    print("\n### Historical Performance Trends ###")
    print(f"Scores from last 5 quizzes: {insights['historical']['trend']}")

    # Difficulty Performance Over Time
    print("\nHistorical Difficulty Performance:")
    for difficulty, score in insights['historical']['difficulty_performance'].items():
        print(f"  - {difficulty}: {score * 100:.2f}%")

    # Personalized Recommendations
    print("\n### Personalized Recommendations ###")
    for rec in recommendations:
        print(f"  - {rec}")

    # Performance Analysis for Focused Improvement
    print("\n### Focused Improvement Areas ###")

    # Identify weak topics
    weak_topics = [topic for topic, score in insights['current']['topic_performance'].items() if score < 60]
    if weak_topics:
        print(f"  - Weak Topics: {', '.join(weak_topics)}. Suggested action: Focus on these topics to improve performance.")
    else:
        print("  - No major weak topics identified.")

    # Focus on Hard Difficulty
    if insights['current']['difficulty_performance']['Hard'] < 0.70:
        print("  - Focus on improving your performance on Hard difficulty questions. This can help boost overall performance.")

    # Time Management Insights
    avg_time = insights['current']['time_management']['avg_time']
    print("\n### Time Management Insights ###")
    if avg_time > 60:
        print("  - You are spending a lot of time per question. Consider working on time management strategies.")
    else:
        print("  - Your time per question is within a reasonable range. Keep practicing!")

# Example Input Data (same as the one you provided)
insights = {
    'current': {
        'overall_score': 70.0,
        'topic_performance': {'Physics': 50.0, 'Chemistry': 100.0, 'Biology': 66.67},
        'difficulty_performance': {'Easy': 0.73, 'Medium': 0.85, 'Hard': 0.67},
        'time_management': {'avg_time': 45.5, 'max_time': 65, 'min_time': 30}
    },
    'historical': {
        'trend': [90.0, 70.0, 40.0, 90.0, 90.0],
        'difficulty_performance': {'Easy': 0.73, 'Medium': 0.85, 'Hard': 0.67}
    }
```

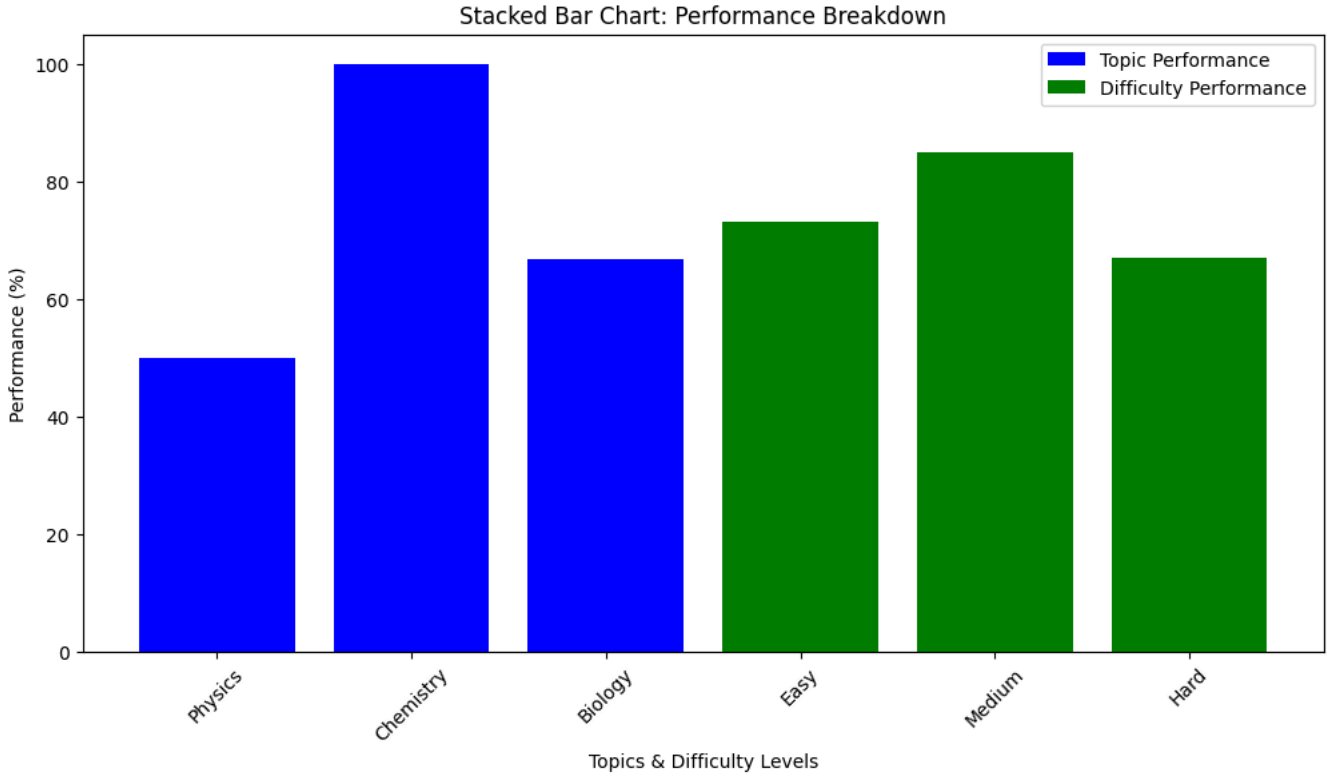
```
}

recommendations = [
    "Focus on improving Physics. Work on key topics like Mechanics and Optics.",
    "Increase practice in Biology, specifically Genetics and Plant Biology.",
    "Work on harder difficulty questions to improve accuracy in challenging problems.",
    "Practice time management through time-bound mock tests to improve speed."
]

# Run the Advanced Output function
generate_advanced_output(insights, recommendations)
```

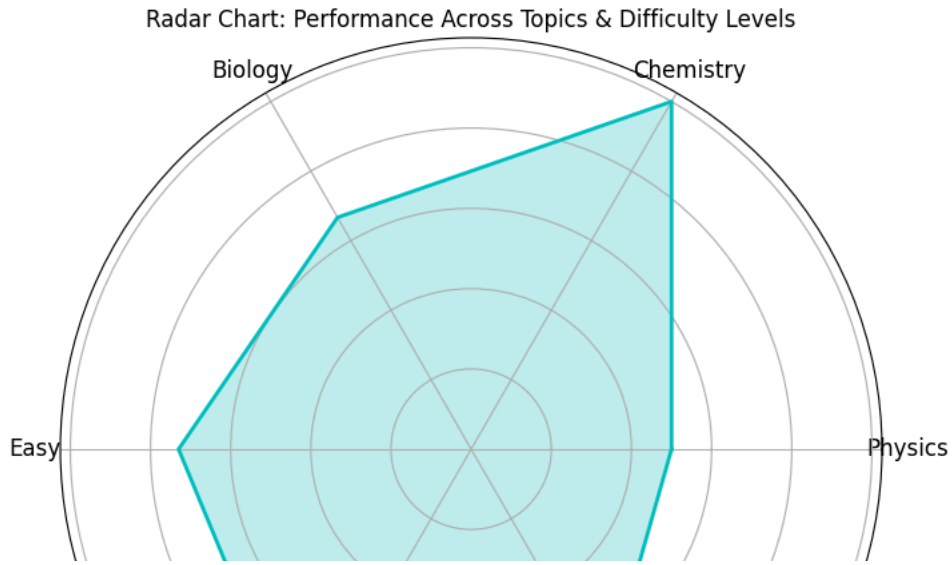
```
### Performance Insights ###
Overall Score: 70.0%

Topic-Wise Performance:
  - Physics: 50.0%
  - Chemistry: 100.0%
  - Biology: 66.67%

Difficulty-Level Performance:
  - Easy: 73.00%
  - Medium: 85.00%
  - Hard: 67.00%

### Historical Performance Trends ###
Scores from last 5 quizzes: [90.0, 70.0, 40.0, 90.0, 90.0]

Historical Difficulty Performance:
  - Easy: 73.00%
  - Medium: 85.00%
  - Hard: 67.00%

### Personalized Recommendations ###
  - Focus on improving Physics. Work on key topics like Mechanics and Optics.
  - Increase practice in Biology, specifically Genetics and Plant Biology.
  - Work on harder difficulty questions to improve accuracy in challenging problems.
  - Practice time management through time-bound mock tests to improve speed.

### Focused Improvement Areas ###
  - Weak Topics: Physics. Suggested action: Focus on these topics to improve performance.
  - Focus on improving your performance on Hard difficulty questions. This can help boost overall performance.

### Time Management Insights ###
  - Your time per question is within a reasonable range. Keep practicing!
```

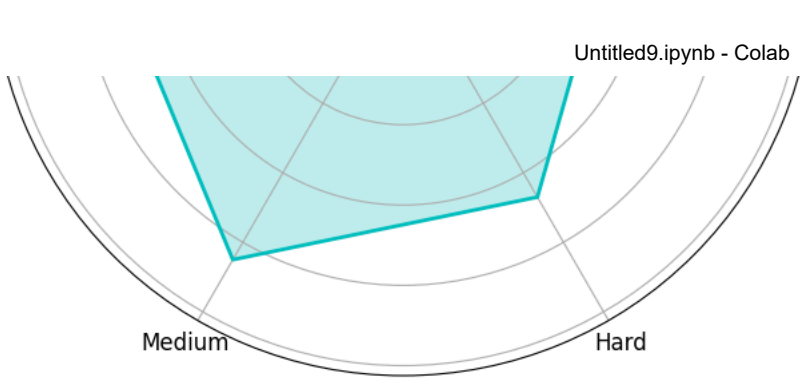
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px

def generate_advanced_visualizations(insights, recommendations):
    # 1. Bubble Chart: Topic Performance vs Difficulty Levels (Bubble Size = Score)
    topics = list(insights['current']['topic_performance'].keys())
    scores = list(insights['current']['topic_performance'].values())
    difficulties = ['Easy', 'Medium', 'Hard']
    difficulty_scores = list(insights['current']['difficulty_performance'].values())

    fig, ax = plt.subplots(1, 1, figsize=(10, 6))
    ax.scatter(topics, scores, s=np.array(difficulty_scores)*1000, alpha=0.5, c='r', label='Topic Performance')
    ax.set_title('Bubble Chart: Topic Performance vs Difficulty Levels')
    ax.set_xlabel('Topics')
    ax.set_ylabel('Performance Score (%)')
    ax.legend(loc='upper right')
    plt.show()

    # 2. Stacked Bar Chart: Topic and Difficulty Performance Breakdown
    topics_performance = insights['current']['topic_performance']
    difficulty_performance = insights['current']['difficulty_performance']

    # Stacked bar data preparation
    categories = list(topics_performance.keys()) + ['Easy', 'Medium', 'Hard']
    performance = list(topics_performance.values()) + [difficulty_performance[d] * 100 for d in difficulties]

    fig, ax = plt.subplots(1, 1, figsize=(12, 6))
    ax.bar(categories[:len(topics_performance)], performance[:len(topics_performance)], label='Topic Performance', color='b')
    ax.bar(categories[len(topics_performance):], performance[len(topics_performance):], label='Difficulty Performance', color='g')
    ax.set_title('Stacked Bar Chart: Performance Breakdown')
    ax.set_xlabel('Topics & Difficulty Levels')
    ax.set_ylabel('Performance (%)')
    ax.legend()
    plt.xticks(rotation=45)
    plt.show()
```

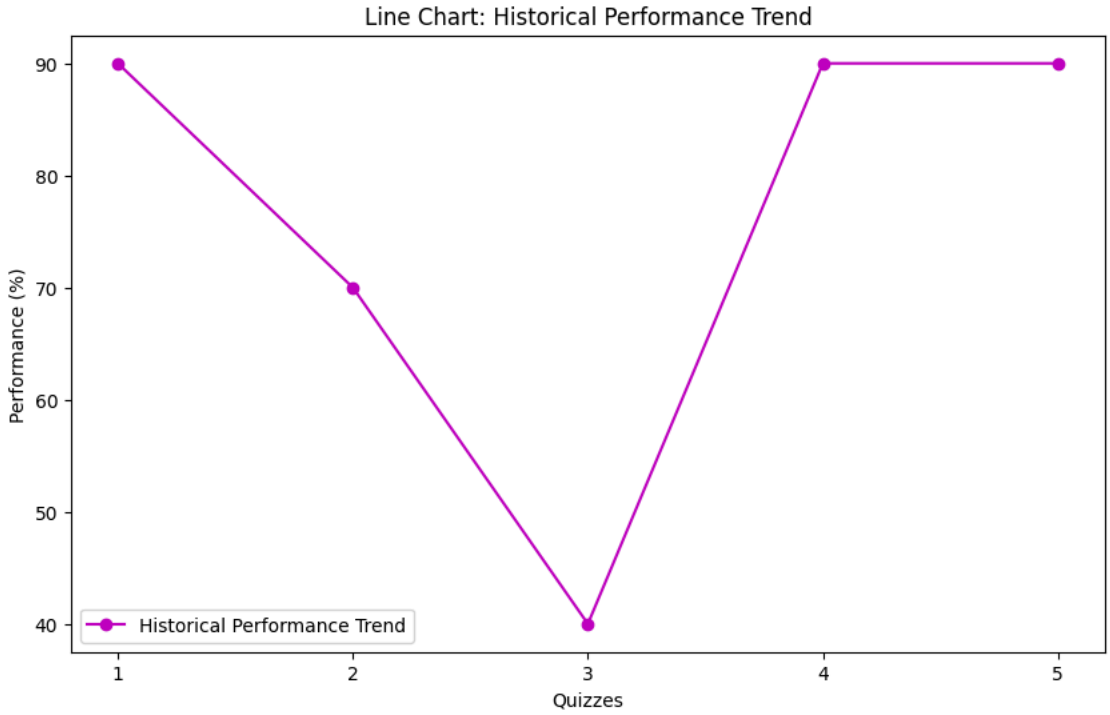
```
    # 3. Radar Chart: Comparing Performance Across Topics and Difficulty Levels
    labels = list(topics_performance.keys()) + difficulties
    values = list(topics_performance.values()) + [difficulty_performance[d] * 100 for d in difficulties]

    # Radar chart setup
    angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
    values += values[:1]  # Complete the loop
    angles += angles[:1]

    fig, ax = plt.subplots(1, 1, figsize=(8, 8), subplot_kw=dict(polar=True))
    ax.fill(angles, values, color='c', alpha=0.25)
    ax.plot(angles, values, color='c', linewidth=2)
    ax.set_title('Radar Chart: Performance Across Topics & Difficulty Levels')
    ax.set_yticklabels([])
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(labels, fontsize=12)
    plt.show()

    # 4. Line Chart: Historical Performance Trend
    historical_scores = insights['historical']['trend']

    fig, ax = plt.subplots(1, 1, figsize=(10, 6))
    ax.plot(range(1, len(historical_scores) + 1), historical_scores, marker='o', color='m', label='Historical Performance Trend')
    ax.set_title('Line Chart: Historical Performance Trend')
    ax.set_xlabel('Quizzes')
    ax.set_ylabel('Performance (%)')
    ax.set_xticks(range(1, len(historical_scores) + 1))
    ax.legend()
    plt.show()

    # 5. Treemap: Performance by Topics and Subtopics
    treemap_data = {
        'Category': ['Physics', 'Physics -> Mechanics', 'Physics -> Optics',
                     'Chemistry', 'Chemistry -> Organic', 'Chemistry -> Inorganic',
                     'Biology', 'Biology -> Genetics', 'Biology -> Plant Biology'],
        'Score': [50.0, 70.0, 30.0, 100.0, 85.0, 75.0, 66.67, 80.0, 70.0],
        'Parent': ['root', 'Physics', 'Physics', 'root', 'Chemistry', 'Chemistry', 'root', 'Biology', 'Biology']
    }

    df_treemap = pd.DataFrame(treemap_data)

    # Create the treemap
    fig = px.treemap(df_treemap, path=['Parent', 'Category'], values='Score',
                     color='Score', hover_data=['Score'], color_continuous_scale='RdBu',
                     title='Treemap: Performance by Topics and Subtopics')
    fig.show()


# Example Input Data (same as the one you provided)
insights = {
    'current': {
        'overall_score': 70.0,
        'topic_performance': {'Physics': 50.0, 'Chemistry': 100.0, 'Biology': 66.67},
        'difficulty_performance': {'Easy': 0.73, 'Medium': 0.85, 'Hard': 0.67},
        'time_management': {'avg_time': 45.5, 'max_time': 65, 'min_time': 30}
    },
    'historical': {
        'trend': [90.0, 70.0, 40.0, 90.0, 90.0],
        'difficulty_performance': {'Easy': 0.73, 'Medium': 0.85, 'Hard': 0.67}
    }
}

recommendations = [
    "Focus on improving Physics. Work on key topics like Mechanics and Optics.",
    "Increase practice in Biology, specifically Genetics and Plant Biology.",
    "Work on harder difficulty questions to improve accuracy in challenging problems.",
    "Practice time management through time-bound mock tests to improve speed."
]

# Run the Advanced Visualizations function
generate_advanced_visualizations(insights, recommendations)
```


Bubble Chart: Topic Performance vs Difficulty Levels
Topic Performance
Performance Score (%)
100
90
80
70
60
50
Physics
Chemistry
Biology
Topics


Stacked Bar Chart: Performance Breakdown
Topic Performance
Difficulty Performance
Performance (%)
100
80
60
40
20
0
Physics
Chemistry
Biology
Easy
Medium
Hard
Topics & Difficulty Levels


Radar Chart: Performance Across Topics & Difficulty Levels
Biology
Chemistry
Easy
Physics





Treemap: Performance by Topics and Subtopics