Sections

Description Sample:

- A steep, rocky uphill segment designed to test balance, throttle control, and navigation skills.

User Group Eligible to Create Sections:

- Any rider

Performance Metrics:

1. **Throttle Control:** Consistency of speed throughout the ascent.
2. **Balance Control:** Stability and control as measured by accelerometer and gyroscope data.
3. **Time Taken:** Total time to complete the ascent.
4. **Turn and Path Accuracy:** How effectively the rider navigates turns and maintains their path.

Algorithm for Evaluation:

Data Collection:

Collect GPS, accelerometer, gyroscope, and speed data during the ride.

Noise Filtering and Normalization:

```
 # Apply filtering to reduce noise in sensor data
def filter_data(sensor_data):
    # Example: Kalman filter for smoother data
    filtered_data = kalman_filter(sensor_data)
    return filtered_data
```

Throttle Control Evaluation:

```
 # Evaluate throttle control based on speed consistency (higher is better)
def evaluate_throttle_control(speed_data):
    consistent_speed = max(speed_data) - min(speed_data)
    return max(0, 10 - consistent_speed)  # Higher score is better, max score is 10
```

Turn and Path Accuracy Analysis:

```
 # Analyze yaw rate and path adherence (higher is better)
def evaluate_turn_and_path_accuracy(yaw_data, gps_data, optimal_path):
    yaw_changes = [abs(yaw_data[i] - yaw_data[i-1]) for i in range(1, len(yaw_data))]
    avg_yaw_change = sum(yaw_changes) / len(yaw_changes)

    deviations = [haversine_distance(gps_data[i], optimal_path[i]) for i in range(len(gps_data))]
    avg_deviation = sum(deviations) / len(deviations)

    turn_path_score = max(0, 10 - (avg_yaw_change + avg_deviation))  # Score out of 10
    return turn_path_score
```

Balance Control Assessment:

```
 # Analyze stability using gyroscope and accelerometer data (higher is better)
def evaluate_balance_control(accel_data, gyro_data):
    stability_score = max(0, 10 - sum(abs(accel_data[i]["tilt"] - accel_data[i-1]["tilt"]) for i in range(1,
len(accel_data))))
    return stability_score  # Score out of 10
```

Time Performance:

```
 # Calculate time taken to complete the section (higher is better for shorter times)
def calculate_time_performance(start_time, end_time):
    time_taken = end_time - start_time
    time_score = max(0, 10 - (time_taken / 60))  # Higher score is better, scaled by time
    return time_score
```

Score and Grade System:

```
 # Assign letter grades based on score ranges
def assign_grade(score):
    if score >= 9.0:
        return "A+"
    elif score >= 8.0:
        return "A"
    elif score >= 7.0:
        return "B+"
    elif score >= 6.0:
        return "B"
    elif score >= 5.0:
        return "C"
    elif score >= 4.0:
        return "D"
```

```
    else:
        return "F"

# Calculate points and grades for each performance metric
def calculate_points_and_grades(throttle_score, balance_score, time_score, turn_path_score):
    total_points = throttle_score + balance_score + time_score + turn_path_score
    grades = {
        "Throttle Control": assign_grade(throttle_score),
        "Balance Control": assign_grade(balance_score),
        "Time Performance": assign_grade(time_score),
        "Turn and Path Accuracy": assign_grade(turn_path_score),
        "Total Points": total_points
    }
    return grades
```

Feedback and Suggestions:

```
 # Provide feedback based on scores
def generate_feedback(throttle_score, balance_score, turn_path_score):
    feedback = [ ]
    if throttle_score < 5.0:
        feedback.append("Improve throttle control by maintaining a steady pace.")
    if balance_score < 5.0:
        feedback.append("Work on body positioning to reduce balance instability.")
    if turn_path_score < 5.0:
        feedback.append("Refine steering control to minimize oversteer and improve path accuracy.")
    return feedback
```

`Sweat_points`.

This function now returns both the grade and the corresponding sweat points.

```
def assign_grade(score):
    if score >= 9.0:
        return "A+", 100
    elif score >= 8.0:
        return "A", 60
    elif score >= 7.0:
        return "B+", 30
    elif score >= 6.0:
        return "B", 12
    elif score >= 5.0:
        return "C", 3
```

```
    elif score >= 4.0:
        return "D", 1
    else:
        return "F", 0

# Example usage
score = 7.5
grade, points = assign_grade(score)
print(f"Grade: {grade}, Sweat Points: {points}")
```

Explanation of Changes:

- The function now returns a tuple containing:
    - The **grade** (e.g., `"A+"`, `"B"`, etc.)
    - The **sweat points** associated with each grade:
        - **A+**: 100 points
        - **A**: 60 points
        - **B+**: 30 points
        - **B**: 12 points
        - **C**: 3 points
        - **D**: 1 point
        - **F**: 0 points