Problem 131: Match Ticketing Analysis Tool POC:

• CRUD: Ticket sales.

• analyze_ticket_sales_data(event_id): Analyze ticket sales data to optimize pricing and promotions.

• predict_future_sales_trends(sales_trend_data): Predict future ticket sales trends.

Code:

Here is a Python solution to create a Match Ticketing Analysis Tool Proof of Concept (POC) that addresses the proposed requirements using Object-Oriented Programming (OOP) and Data Structures and Algorithms (DSA) principles. The tool will be designed to handle CRUD operations for ticket sales and to provide functions for analyzing and predicting ticket sales trends.

### Step 1: Define Classes for Ticket Sales

```python
class Ticket:
    def __init__(self, event_id, price, sold=False):
        self.event_id = event_id
        self.price = price
        self.sold = sold


class TicketSales:
    def __init__(self):
        # A dictionary to store ticket information with event_id as key
        # Each event_id maps to a list of Ticket instances
        self.sales = {}


    def add_ticket(self, event_id, price):
        ticket = Ticket(event_id, price)
        if event_id not in self.sales:
            self.sales[event_id] = []
        self.sales[event_id].append(ticket)

    def sell_ticket(self, event_id):
        for ticket in self.sales.get(event_id, []):
            if not ticket.sold:
                ticket.sold = True
                return True
        return False  # No tickets available to sell

    def delete_ticket(self, event_id, ticket_index):
        if event_id in self.sales and 0 <= ticket_index < len(self.sales[event_id]):
            del self.sales[event_id][ticket_index]
            return True
        return False

    def get_tickets(self, event_id):
        return self.sales.get(event_id, [])
```

### Step 2: Analysis and Prediction Functions

```python
def analyze_ticket_sales_data(ticket_sales, event_id):
    tickets = ticket_sales.get_tickets(event_id)
    if not tickets:
        return "No tickets data available."

    # Analysis: average price, percentage sold
    total_sold = sum(1 for t in tickets if t.sold)
    total_tickets = len(tickets)
    average_price = sum(t.price for t in tickets) / total_tickets
    sold_percentage = (total_sold / total_tickets) * 100

    return {
        "average_price": average_price,
        "sold_percentage": sold_percentage,
        "total_sold": total_sold,
        "total_tickets": total_tickets
    }

def predict_future_sales_trends(sales_trend_data):
    # Dummy model for prediction
    trends = {"increase": 0, "decrease": 0}
    for data in sales_trend_data:
        if data["sold_trend"] > 0:
            trends["increase"] += 1
        else:
            trends["decrease"] += 1

    return trends
```

### Step 3: Unit Testing with Unittest Framework

```python
import unittest
```

```
class TestTicketSales(unittest.TestCase):
    def setUp(self):
        self.ticket_sales = TicketSales()
        self.event_id = "E001"
        self.ticket_sales.add_ticket(self.event_id, 100)

    def test_add_and_sell_ticket(self):
        self.ticket_sales.add_ticket(self.event_id, 150)
        self.assertTrue(self.ticket_sales.sell_ticket(self.event_id))
        self.assertEqual(len(self.ticket_sales.get_tickets(self.event_id)), 2)
        self.assertTrue(self.ticket_sales.sell_ticket(self.event_id))
        self.assertFalse(self.ticket_sales.sell_ticket(self.event_id))  # All tickets sold

    def test_ticket_deletion(self):
        self.assertTrue(self.ticket_sales.delete_ticket(self.event_id, 0))
        self.assertFalse(self.ticket_sales.delete_ticket(self.event_id, 0))

    def test_analyze_sales(self):
        self.ticket_sales.add_ticket(self.event_id, 150)
        self.ticket_sales.sell_ticket(self.event_id)
        analysis = analyze_ticket_sales_data(self.ticket_sales, self.event_id)
        self.assertEqual(analysis['total_sold'], 1)

class TestSalesPrediction(unittest.TestCase):
    def test_prediction(self):
        trend_data = [{"sold_trend": 1}, {"sold_trend": -1}, {"sold_trend": 1}]
        prediction = predict_future_sales_trends(trend_data)
        self.assertEqual(prediction["increase"], 2)
        self.assertEqual(prediction["decrease"], 1)
```

```
if __name__ == '__main__':
    unittest.main()
```

### Execution

The solution can be run by copying the classes and functions defined along with the unit tests into a Python file and executing it. Ensure all dependencies such as Python's `unittest` library are properly installed. The tool handles CRUD operations for ticket sales and provides fundamental analysis of sales data. Prediction logic in the POC just counts trends; for a full solution, integrate a statistical model or machine learning algorithm.