1. Python Coding Challenge: Decision Tree for Classification

```
from collections import Counter

class Node:
  def __init__(self, feature_index, threshold, left_child=None, right_child=None, label=None):
    self.feature_index = feature_index
    self.threshold = threshold
    self.left_child = left_child
    self.right_child = right_child
    self.label = label

def calculate_gini(data):
  labels = [row[-1] for row in data]
  label_counts = Counter(labels)
  total = len(labels)
  gini = 1 - sum((count / total) ** 2 for count in label_counts.values())
  return gini

def find_best_split(data):
  best_gini = float("inf")
  best_feature_index = None
  best_threshold = None
  for feature_index in range(len(data[0]) - 1):
    feature_values = [row[feature_index] for row in data]
    unique_values = set(feature_values)
    for threshold in unique_values:
      left_data = [row for row in data if row[feature_index] <= threshold]
      right_data = [row for row in data if row[feature_index] > threshold]
      gini = calculate_gini(left_data) + calculate_gini(right_data)
      if gini < best_gini:
        best_gini = gini
        best_feature_index = feature_index
        best_threshold = threshold
  return best_feature_index, best_threshold

def build_tree(data):
  if len(data) == 0:
    return None
  if all(label == data[0][-1] for label in [row[-1] for row in data]):
    return Node(None, None, label=data[0][-1])
  feature_index, threshold = find_best_split(data)
  left_data = [row for row in data if row[feature_index] <= threshold]
  right_data = [row for row in data if row[feature_index] > threshold]
  left_child = build_tree(left_data)
  right_child = build_tree(right_data)
  return Node(feature_index, threshold, left_child, right_child)

def predict(node, instance):
  if node.label is not None:
    return node.label
  if instance[node.feature_index] <= node.threshold:
    return predict(node.left_child, instance)
  else:
    return predict(node.right_child, instance)

# Example usage (replace with your actual dataset)
data = [
  [1, 2, "yes"],
  [2, 1, "no"],
  [3, 3, "yes"],
  [4, 2, "no"],
  [5, 1, "yes"],
]

tree = build_tree(data)
prediction = predict(tree, [6, 3])
print(f"Prediction: {prediction}")

# Evaluate performance using metrics like accuracy, precision, recall, etc.
```

This code defines a Node class to represent decision tree nodes and functions for calculating Gini impurity, finding the best split, building the tree, and making predictions. You'll need to replace the example data with your actual dataset and implement appropriate performance evaluation metrics.

2. Deep Learning Questions:

a) Shallow vs. Deep Neural Networks:

- **Shallow networks: Fewer hidden layers (typically 1-3), suitable for simpler tasks, faster to train.**
- **Deep networks: More hidden layers, powerful for complex relationships, computationally expensive, prone to overfitting.**

b) Backpropagation:

- **Algorithm to train deep neural networks.**
- **Adjusts network weights to minimize the loss between predictions and actual values.**
- **Uses gradient descent to iteratively update weights in the opposite direction of the loss gradient.**
- **Enables learning from mistakes and improvement over time.**

3. Classical Machine Learning Questions:

a) Overfitting and Underfitting:

- **Overfitting: Model memorizes training data too well, failing to generalize to unseen data.**

- **Underfitting: Model fails to capture the underlying patterns in the data, resulting in poor performance on both training and testing data.**

b) Addressing Overfitting and Underfitting:

- **Regularization: Techniques like L1/L2 regularization penalize large weights, preventing overfitting.**
- **Data augmentation: Artificially increasing training data diversity to improve generalization.**
- **Early stopping: Halting training when validation performance starts to deteriorate to prevent overfitting.**
- **Choosing the right model complexity: Selecting a model with appropriate capacity to avoid underfitting or overfitting.**

4. Algorithm Building for Anomaly Detection:

a) Anomaly Detection Algorithm:

1. **Define normal behaviour: Analyse historical data to establish a baseline for normal patterns.**
2. **Choose an anomaly detection technique:**
   - **Statistical methods: Identify data points deviating significantly from statistical properties (e.g., mean, standard deviation).**
   - **Clustering algorithms: Group similar data points, identifying outliers that fall outside clusters.**
   - **Machine learning models: Train models to distinguish normal from anomalous data.**
3. **Evaluate and refine: Assess the algorithm's performance and adjust parameters or techniques as needed.**

b) Key Steps:

- **Data preprocessing: Cleaning, handling missing values, scaling features.**
- **Feature engineering: Creating new features potentially useful for anomaly detection.**
- **Model selection and training: Choosing an appropriate anomaly detection technique and training it on labelled or unlabelled data.**
- **Anomaly scoring: Assigning anomaly scores to data points based on their deviation from normality.**
- **Thresholding and decision-making: Setting a threshold to classify data points as normal or anomalous.**

5. Artificial Intelligence (AI):

AI refers to the field of creating intelligent machines capable of performing tasks typically requiring human intelligence. It involves:

- **Simulating human intelligence: Enabling machines to understand, learn, and make decisions.**
- **Subfields: Machine learning, natural language processing, computer vision, robotics, etc.**
- **Goals: Replicating human-like behaviour using algorithms, statistical models, and computational techniques.**