1. Refer the below given dataset.

Outlook Temperature Humidity Wind Played football(yes/no)

Sunny Hot High Weak No

Sunny Hot High Strong No

Overcast Hot High Weak Yes

Rain Mild High Weak Yes

Rain Cool Normal Weak Yes

Rain Cool Normal Strong No

Overcast Cool Normal Strong Yes

Sunny Mild High Weak No

Sunny Cool Normal Weak Yes

Rain Cool Normal Weak Yes

Sunny Mild Normal Strong Yes

Overcast Mild Normal Strong Yes

Overcast Hot High Strong Yes

Rain Mild High Strong No

Create a decision tree from scratch using the above dataset using ID3 algorithm.

Given a new set of features, predict whether game will be played.

Outlook Temperature Humidity Wind Play

Rain Hot Normal Weak ?

```
# Load necessary libraries
import pandas as pd
import numpy as np

# Load the dataset
data = pd.DataFrame({
    'Outlook': ['Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Sunny',
'Overcast', 'Rain', 'Rain', 'Sunny', 'Rain', 'Overcast', 'Overcast',
'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
```

```
'Strong'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})

# Define a function to calculate the entropy of a dataset
def entropy(data):

    # Calculate unique values and their counts
    values, counts = np.unique(data, return_counts=True)

    # Calculate probabilities
    probs = counts / len(data)

    # Calculate entropy using the formula
    entropy = -np.sum(probs * np.log2(probs))
    return entropy

# Define a function to calculate the information gain of a feature
def information_gain(data, feature):

    # Initialize feature_entropy
    feature_entropy = 0

    # Get unique values of the feature
    values = data[feature].unique()
    for value in values:

        # Subset the data based on the feature value
        subset = data[data[feature] == value]

        # Calculate subset entropy
        subset_entropy = entropy(subset['Play'])

        # Calculate weight and add to feature_entropy
        weight = len(subset) / len(data)
        feature_entropy += weight * subset_entropy

    # Calculate information gain
    information_gain = entropy(data['Play']) - feature_entropy
    return information_gain

# Define a function to build the decision tree
def build_tree(data, features, target):

    # Base cases
    if len(data[target].unique()) == 1:
        return data[target].iloc[0]
    if len(features) == 0:
        return data[target].value_counts().idxmax()
```

```
    # Choose the feature with the highest information gain
    information_gains = [information_gain(data, feature) for feature
in features]
    best_feature_index = np.argmax(information_gains)
    best_feature = features[best_feature_index]

    # Create a new decision tree node
    tree = {best_feature: {}}

    # Remove the best feature from the feature list
    features = [feature for feature in features if feature !=
best_feature]

    # Recursively build the subtree for each value of the best feature
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        subtree = build_tree(subset, features, target)
        tree[best_feature][value] = subtree
    return tree

# Build the decision tree
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
target = 'Play'
tree = build_tree(data, features, target)

# Define a function to predict the classification of a new example
def predict(example, tree):
    for feature, subtree in tree.items():
        value = example[feature]
        subtree = subtree[value]
        if isinstance(subtree, dict):
            return predict(example, subtree)
        else:
            return subtree

# Example usage
new_example = {'Outlook': 'Sunny', 'Temperature': 'Normal',
'Humidity': 'High', 'Wind': 'Weak'}
prediction = predict(new_example, tree)
print(f'The predicted classification for the new example is
{prediction}.')
```

```
The predicted classification for the new example is Yes.
```

2. For the same above problem create a decision tree using scikit learn API. Predict for the new set of features given as in the question 1.

```
# Load necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.DataFrame({
    'Outlook': ['Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Sunny',
'Overcast', 'Rain', 'Rain', 'Sunny', 'Rain', 'Overcast', 'Overcast',
'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Strong'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})

# Encode the categorical features
encoder = LabelEncoder()
data['Outlook'] = encoder.fit_transform(data['Outlook'])
data['Temperature'] = encoder.fit_transform(data['Temperature'])
data['Humidity'] = encoder.fit_transform(data['Humidity'])
data['Wind'] = encoder.fit_transform(data['Wind'])
data['Play'] = encoder.fit_transform(data['Play'])

# Split the dataset into features and target
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']
target = 'Play'
X = data[features]
y = data[target]

# Build the decision tree
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(X, y)

# Predict the classification of a new example
new_example = [[0, 1, 0, 1]]  # Sunny, Normal, High, Weak
prediction = encoder.inverse_transform(tree.predict(new_example))[0]
```

```
print(f'The predicted classification for the new example is {prediction}.')
```

```
The predicted classification for the new example is Yes.

C:\Users\raosu\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

3. Implement KNN on Social Network Ads.csv. Find the Accuracy of classification.

```
# Import necessary libraries
import pandas as pd
import numpy as np

# Read the CSV file
data = pd.read_csv('C:/Users/raosu/Documents/Assignment 8 aiml/Social_Network_Ads.csv')

# Considering two features for classification (Age and EstimatedSalary)
X = data.iloc[:, [2, 3]].values  # Assuming columns 2 and 3 are 'Age' and 'EstimatedSalary'
y = data.iloc[:, -1].values  # Assuming the last column is the target 'Purchased'

# Feature scaling (standardization)
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Function to calculate Euclidean distance
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

# Function to perform KNN classification
def KNN_predict(X_train, y_train, X_test, k):
    predictions = []
    for test_point in X_test:
        distances = []
        for train_point in X_train:
            dist = euclidean_distance(test_point, train_point)
            distances.append(dist)
        nearest_indices = np.argsort(distances)[:k]  # Indices of k nearest neighbors
        nearest_labels = [y_train[i] for i in nearest_indices]
        predicted_label = max(set(nearest_labels),
```

```
key=nearest_labels.count)
        predictions.append(predicted_label)
    return predictions

# Splitting the data into training and test sets (80% train, 20% test)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Define the value of K
k = 5

# Make predictions
predictions = KNN_predict(X_train, y_train, X_test, k)

# Calculate accuracy
correct = sum(predictions[i] == y_test[i] for i in
range(len(predictions)))
accuracy = correct / len(predictions)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9000
```