Nr. 16

a) Satz von Bayes beweisen:

$$P(S|W) = \frac{P(W|S) \cdot P(S)}{P(W)}$$

bedingte Wahrscheinlichkeit:

$$P(S|W) = \frac{P(S \cap W)}{P(W)} \qquad P(W) > 0$$

$$P(W|S) = \frac{P(S \cap W)}{P(S)}$$

$$P(S \cap W) = P(S|W) \cdot P(W) = P(W|S) \cdot P(S)$$

$$\Rightarrow P(S|W) = \frac{P(W|S) \cdot P(S)}{P(W)}$$ ✓

$\square$

b) Wahrscheinlichkeit, dass heute Fußball gespielt wird:

Attribute	Value	S = yes	S = No
Wind	high	3	3
Humidity	high	3	4
Temperature	cold	3	3
Forecast	Sunny	2	1

14 Messungen $\Rightarrow$ $P(S = yes) = \frac{9}{14}$

$P(S = NO) = \frac{5}{14}$

} Wahrscheinlichkeit für jede einzelne Wetterbedingung, dass Fußball gespielt wird

$$P(W|S) = \prod_i (P(X_i|S))$$

$$P(W|S = yes) = \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{2}{9} = \frac{2}{243}$$ ✓

$$P(W|S=NO) = \frac{3}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{1}{5} = \frac{36}{625}$$ ✓

Norming $P(W)$: totale Wahrscheinlichkeit

$$P(W) = P(W|S=yes) \cdot P(S=yes) + P(W|S=NO) \cdot P(S=No)$$

$$= \frac{2}{243} \cdot \frac{5}{14} + \frac{36}{625} \cdot \frac{5}{14} = \frac{611}{23625}$$ ✓

$$\Rightarrow P(S=yes|W) = \frac{P(W|S=yes) \cdot P(S)}{P(W)}$$

$$= \frac{\frac{2}{243} \cdot \frac{9}{14}}{\frac{611}{23625}} = \frac{125}{611} = 0{,}2 = 20\%$$

Mit einer Wahrscheinlichkeit von 20% wird heute Fußball gespielt.

c) Wahrscheinlichkeit, dass morgen Fußball gespielt wird

Attribute	Value	S=yes	S=No
Wind	low	6	2
Humidity	high	3	4
Temperature	hot	0	1
Forecast	Sunny	2	1

c)

We read in the data, same as in b).

Notably, there are no games played on a hot day. This results in a probabiliy of `zero`, which raises an issue.

In particular, the value `p_temp` is used to calculate the overall probability in

```
p_W = (p_W_yes * p_S) + (p_W_no * p_S_)
```

```
p_SW = p_W_yes * p_S / p_W
```

, so it appears in both, the normalization and in the final scaling of `p_SW`.

One possible workaround is the **Add-One Smoothing** or **Laplace Smoothing**. This methods adds an arbitrary 1 to each attribute count. (To mitigate those counts which are 0) At the same time, the overall counter is increased by the total number of attributes; to account for the 1-addition.

Yes, possible and that is what was wanted,
I am not a fan of this solution.
Discussion -> see class

In formulas, this method can be described as follows:

$$p_{i,\text{ empirical}} = \frac{x_i}{N}$$

This is the problematic probability, being zero. We smooth it out by adding

$$p_{i,\alpha\text{-smoothed}} = \frac{x_i + \alpha}{N + \alpha d}$$

as if to increase each count x_i by α a priori. (In this case $\alpha = 1$, $d = 4$)

In [ ]:
```
import pandas as pd

df_c = pd.DataFrame(
        [["Wind", "low", 6, 2],
        ["Humidity", "high", 3, 4],
        ["Temperature", "hot", 0, 1],
        ["Forecast", "sunny", 2, 1]],
    columns=["attribute", "value", "S=yes", "S=no"]
)
df_c.head()

sum_yes = 9 + 4
# Calculate probabilities with Laplace smoothing (instead of omitting values)
p_wind = (df_c.loc[0, "S=yes"] + 1) / sum_yes
p_humidity = (df_c.loc[1, "S=yes"] + 1) / sum_yes
p_temp = (df_c.loc[2, "S=yes"] + 1) / sum_yes
p_cast = (df_c.loc[3, "S=yes"] + 1) / sum_yes

p_W_yes = p_wind * p_humidity * p_temp * p_cast

sum_no = 5 + 4
p_wind_no = (df_c.loc[0, "S=no"] + 1) / sum_no
p_humidity_no = (df_c.loc[1, "S=no"] + 1) / sum_no
p_temp_no = (df_c.loc[2, "S=no"] + 1) / sum_no
p_cast_no = (df_c.loc[3, "S=no"] + 1) / sum_no

p_W_no = p_wind_no * p_humidity_no * p_temp_no * p_cast_no

total = sum_yes + sum_no
p_S = sum_yes / total
p_S_ = sum_no / total

p_W = (p_W_yes * p_S) + (p_W_no * p_S_)

p_SW = p_W_yes * p_S / p_W

print(f"Probability for today: {p_SW:.4f} %")
```

```
Probability for today: 0.3172 %
```

As reference: The probability with omitting of `p_temp` is ~58%.

Nr. 17

a) Calculate the entropy of the tree's root

$$H(Y) = - \sum_{z \in Z} P(Y=z) \log_2 P(Y=z)$$ → Entropie before a split

- random variable Y

→ takes values from finite set of symbols Z

$$H(Y|X) = \sum_{m \in M} P(X=m)\, H(Y|X=m)$$ → Entropy after the split

$$= - \sum_{m \in M} P(X=m) \sum_{z \in Z} P(Y=z|X=m) \log P(Y=z \mid X=m)$$

⇒ $N_{Messung} = 14$

True = 9

False = 5

$$H(Y) = H(S) = - \sum_{z \in \{True, False\}} P(Socar = z) \log_2 (P(Socar = z))$$

$$= - \left[P(S = True) \cdot \log_2 (P(S = True)) + P(S = False) \cdot \log_2 (P(S = False)) \right]$$

$$P(S = True) = \frac{9}{14}$$

$$P(S = False) = \frac{5}{14}$$

$$H(S) = - \left[\frac{9}{14} \cdot \log_2 \left(\frac{9}{14} \right) + \frac{5}{14} \cdot \log_2 \left(\frac{5}{14} \right) \right] = 0{,}94$$ ✓

b) Calculate the information gain if cut is made on the attribute wind

$IG(X, Y) = (H(Y) - H(Y|X))$

wind:
6 true
8 false

$X =$ wind
$Y =$ Soccer

$$H(Y|X) = \sum_{m \in M} P(X=m)\, H(Y|X=m)$$

$$= \sum_{m \in \{true, false\}}$$

$$H(S| \text{Wind}=\text{True}) = -\Big[P(S=\text{true} \mid \text{Wind}=\text{true}) \cdot \log_2\big(P(S=\text{true} \mid \text{Wind}=\text{true})\big)$$
$$+ P(S=\text{false} \mid \text{Wind}=\text{true}) \cdot \log_2\big(P(S=\text{false} \mid \text{wind}=\text{true})\big)\Big]$$
$$= -\Big[\tfrac{3}{6} \cdot \log_2\big(\tfrac{3}{6}\big) + \tfrac{3}{6} \cdot \log_2\big(\tfrac{3}{6}\big)\Big] = 1$$

$$H(S| \text{wind}=\text{false}) = -\Big[\tfrac{6}{8} \log_2\big(\tfrac{6}{8}\big) + \tfrac{2}{8} \log_2\big(\tfrac{2}{8}\big)\Big] = 0{,}811 \checkmark$$

$$H(S|\text{Wind}) = P(\text{Wind}=\text{true}) \cdot H(S| \text{wind}=\text{true}) + P(\text{wind}=\text{false}) \cdot H(S| \text{wind}=\text{false})$$
$$= \tfrac{6}{14} \cdot 1 + \tfrac{8}{14} \cdot 0{,}811 = 0{,}892 \checkmark$$

$$IG(\text{Wind}, S) = H(S) - H(S|\text{Wind})$$
$$= 0{,}94 - 0{,}892 = 0{,}048 \checkmark$$

$$\sum = 5/5$$
$A16$

In [ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("soccer.csv")

d = {'sunny': 0, 'cloudy': 1, 'rainy': 2}
inverse_d = {v: k for k, v in d.items()}    # For later use
df["forecast_code"] = df["weather_forecast"].map(d)

def entropy(target: str, df: pd.DataFrame) -> float:
    '''
    The type of the attribute doesn't matter here.
    The cut is done beforehand and this function only handles the subset of the cut.
    The distinction between `soccer==True` and `False` is made in all subsets.
    '''
    count_yes = len(df[df[target] == True])
    count_no = len(df[df[target] == False])

    count_sum = count_yes + count_no

    p_yes = count_yes / count_sum
    p_no = count_no / count_sum

    H = - (p_yes * np.log2(p_yes) if p_yes > 0 else 0) - (p_no * np.log2(p_no) if p_no > 0 else 0)

    return H

def information_gain(target: str, df: pd.DataFrame, attribute: str, type: str) -> float:
    '''
    Parameters
    ----------
    >>> type: ["ordinal", "nominal", "cardinal"]
    ordinal:  Ordered, non-numerical values (e.g. low, medium, high).
    nominal:  Non-ordered, non-numerical, categorical values (sunny, rainy, cloudy).
    cardinal: Continuous, numerical values (e.g. temperature = 17.8, 21.1, ...).
    '''

    # Initialization
    # Total entropy
    total_entropy = entropy(target, df)

    total_samples = len(df)
    weighted_entropy = 0
    best_cut = None

    if (type == "nominal"):
        # The case of nominal values could be done with the One-Hot-Encoding
        # This implementation however iterates over every possible combination of classes.
        # This option is considered viable for a small attribute value count. (in this case 2 and 3)
        unique_values = df[attribute].unique()
        min_weighted_entropy  = 1.

        plot_values = []

        for i in unique_values:
            cut_value = i
            left_sub_df = df[df[attribute] == cut_value]
            right_sub_df = df[df[attribute] != cut_value]
            left_sub_entropy = entropy(target, left_sub_df)
            right_sub_entropy = entropy(target, right_sub_df)

            weighted_entropy = (len(left_sub_df) / total_samples * left_sub_entropy +
                                len(right_sub_df) / total_samples * right_sub_entropy)

            if weighted_entropy < min_weighted_entropy:
                min_weighted_entropy = weighted_entropy
                best_cut = cut_value

            plot_values.append((cut_value, total_entropy - weighted_entropy))

        # Create plot
        plt.figure(figsize=(10,5))
        plt.bar([str(val[0]) for val in plot_values], [val[1] for val in plot_values])
        plt.xlabel('Cut')
        plt.ylabel('Information Gain')
```

```
        plt.title(f'Information Gain for different cuts of attribute {attribute}')
        plt.show()

        weighted_entropy = min_weighted_entropy

    elif (type == "ordinal"):
        # We don't have this case, so there's no need to implement it.
        print("No ordinal values in data set.")
    elif (type == "cardinal"):
        # Do several cuts and determine the lowest weighted_entropy (to maximize the information gain)
        # Determine the cut by taking the mean between two values. For n values, we have n-1 possible cuts.
        df = df.sort_values(by=attribute)
        min_weighted_entropy  = 1

        plot_values = []

        for i in range(total_samples - 1):
            if (df.iloc[i].at[attribute] == df.iloc[i+1].at[attribute]):
                continue
            cut_value = (df.iloc[i].at[attribute] + df.iloc[i+1].at[attribute]) / 2
            left_sub_df = df[df[attribute] <= cut_value]
            right_sub_df = df[df[attribute] > cut_value]
            left_sub_entropy = entropy(target, left_sub_df)
            right_sub_entropy = entropy(target, right_sub_df)

            if (len(left_sub_df) == 0 or len(right_sub_df) == 0):
                print(f"Null subset at {df.loc[i].at[attribute]}, i:{i}")

            weighted_entropy = (len(left_sub_df) / total_samples * left_sub_entropy +
                                len(right_sub_df) / total_samples * right_sub_entropy)

            if weighted_entropy < min_weighted_entropy:
                min_weighted_entropy = weighted_entropy
                best_cut = cut_value

            plot_values.append((cut_value, total_entropy - weighted_entropy))

        # Create plot
        plt.figure(figsize=(10,5))
        plt.bar([val[0] for val in plot_values], [val[1] for val in plot_values])
        plt.xlabel('Cut')
        plt.ylabel('Information Gain')
        plt.title(f'Information Gain for different cuts of attribute {attribute}')
        plt.show()

        weighted_entropy = min_weighted_entropy



    return best_cut, total_entropy - weighted_entropy


df = df.sort_values("temperature", ascending=False)
df
```

Out[ ]:

	temperature	weather_forecast	humidity	wind	soccer	forecast_code
0	29.4	sunny	85	False	False	0
2	28.3	cloudy	78	False	True	1
12	27.2	cloudy	75	False	True	1
1	26.7	sunny	90	True	False	0
9	23.9	rainy	80	False	True	2
10	23.9	sunny	70	True	True	0
7	22.2	sunny	95	False	False	0
11	22.2	cloudy	90	True	True	1
13	21.7	rainy	80	True	False	2
3	21.1	rainy	96	False	True	2
8	20.6	sunny	70	False	True	0
4	20.0	rainy	80	False	True	2
5	18.3	rainy	70	True	False	2
6	17.8	cloudy	65	True	True	1

a)

$H(Y) = -\sum_{z \in Z} P(Y=z) \log_2 P(Y=z)$

In [ ]:

```
H_root = entropy("soccer", df)
print(f"Entropy root: {H_root}\n")
```

```
Entropy root: 0.9402859586706311
```

b)

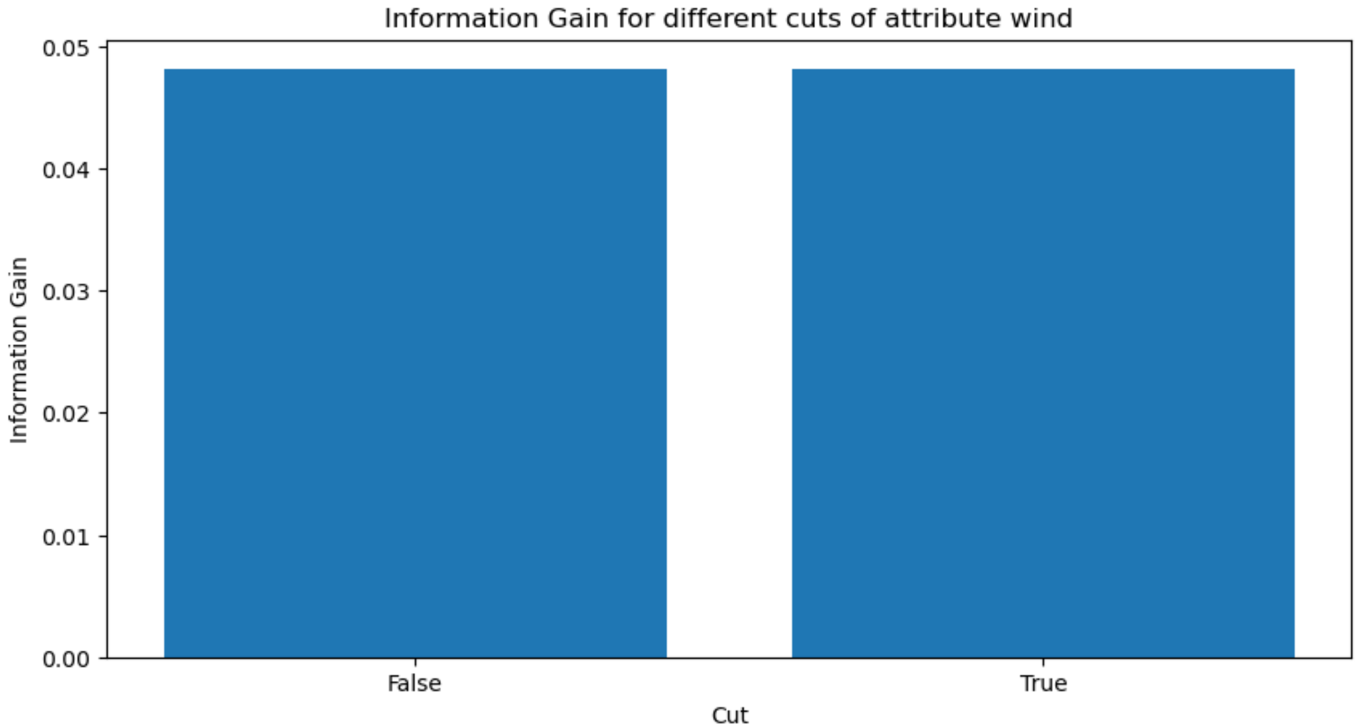
Information gain: $IG(X, Y) = H(Y) - H(Y \mid X)$

with

$$
\begin{aligned}
H(Y \mid X) &= \sum_{m \in M} P(X=m) H(Y \mid X=m) \\
&= -\sum_{m \in M} P(X=m) \sum_{z \in Z} P(Y=z \mid X=m) \log P(Y=z \mid X=m)
\end{aligned}
$$

In [ ]:

```
cut_wind, information_gain_wind = information_gain("soccer", df, "wind", "nominal")
print(f"Information gain wind: {information_gain_wind}\n")
```

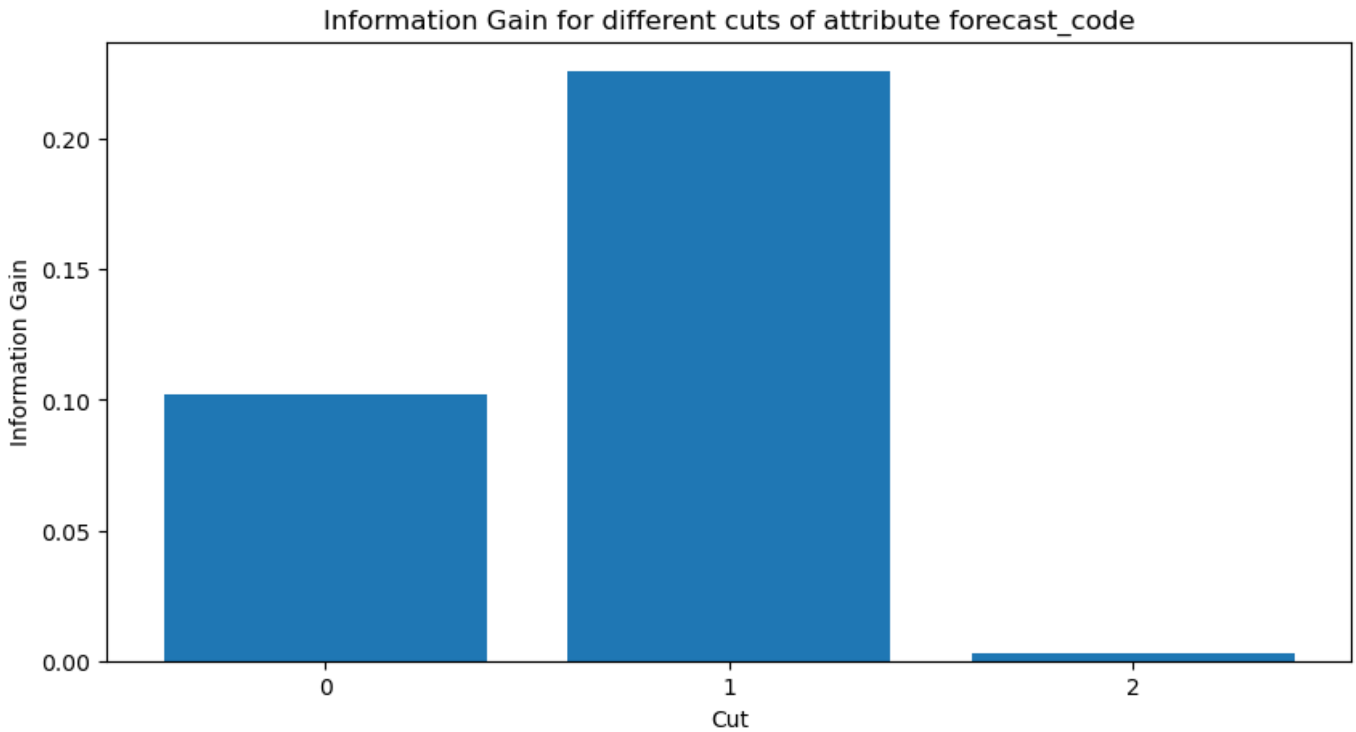



```
Information gain wind: 0.04812703040826949
```

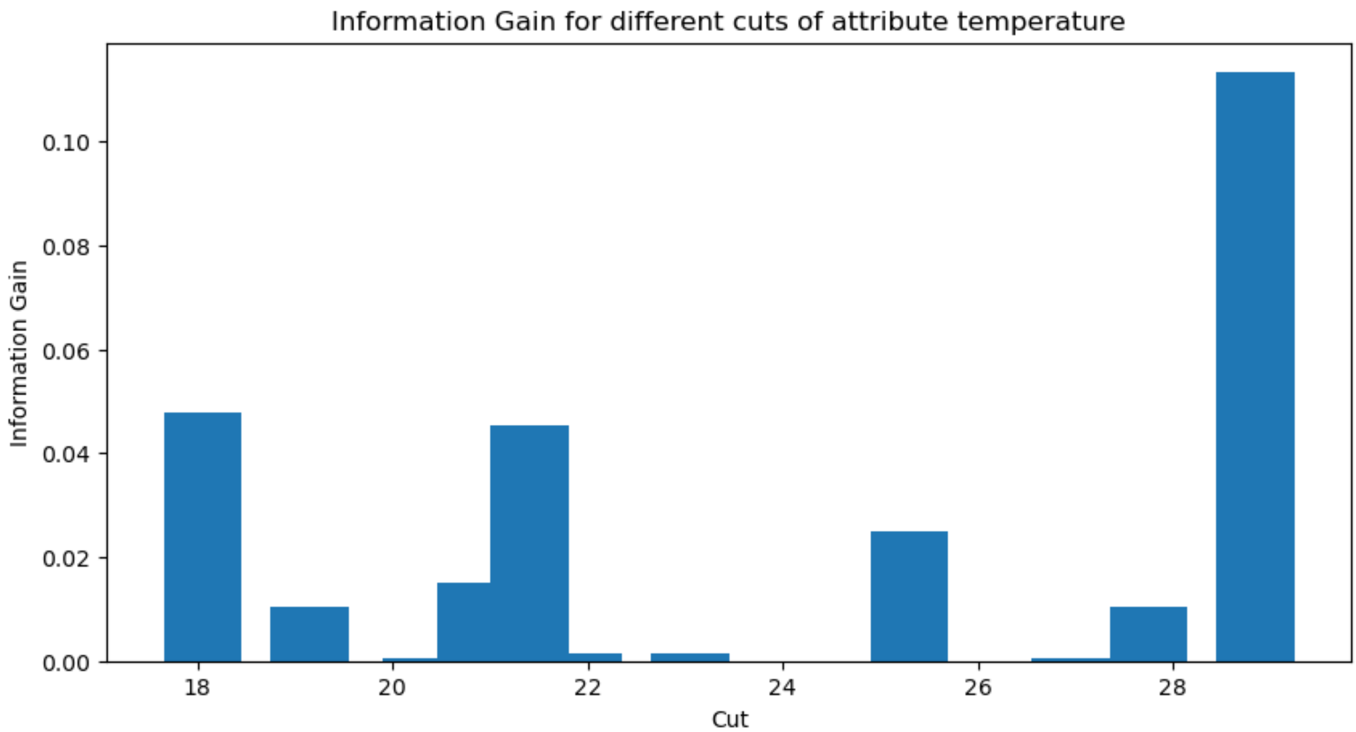
c)

```
In [ ]: cut_forecast, information_gain_forecast = information_gain("soccer", df, "forecast_code", "nominal")
print(f"Information gain forecast: {information_gain_forecast}")
print(f"Best cut forecast: {inverse_d[cut_forecast]}\n")

cut_temperature, information_gain_temperature = information_gain("soccer", df, "temperature", "cardinal")
print(f"Information gain temperature: {information_gain_temperature}")
print(f"Best cut temperature: {cut_temperature}\n")

cut_humidity, information_gain_humidity = information_gain("soccer", df, "humidity", "cardinal")
print(f"Information gain humidity: {information_gain_humidity}")
print(f"Best cut humidity: {cut_humidity}\n")
```

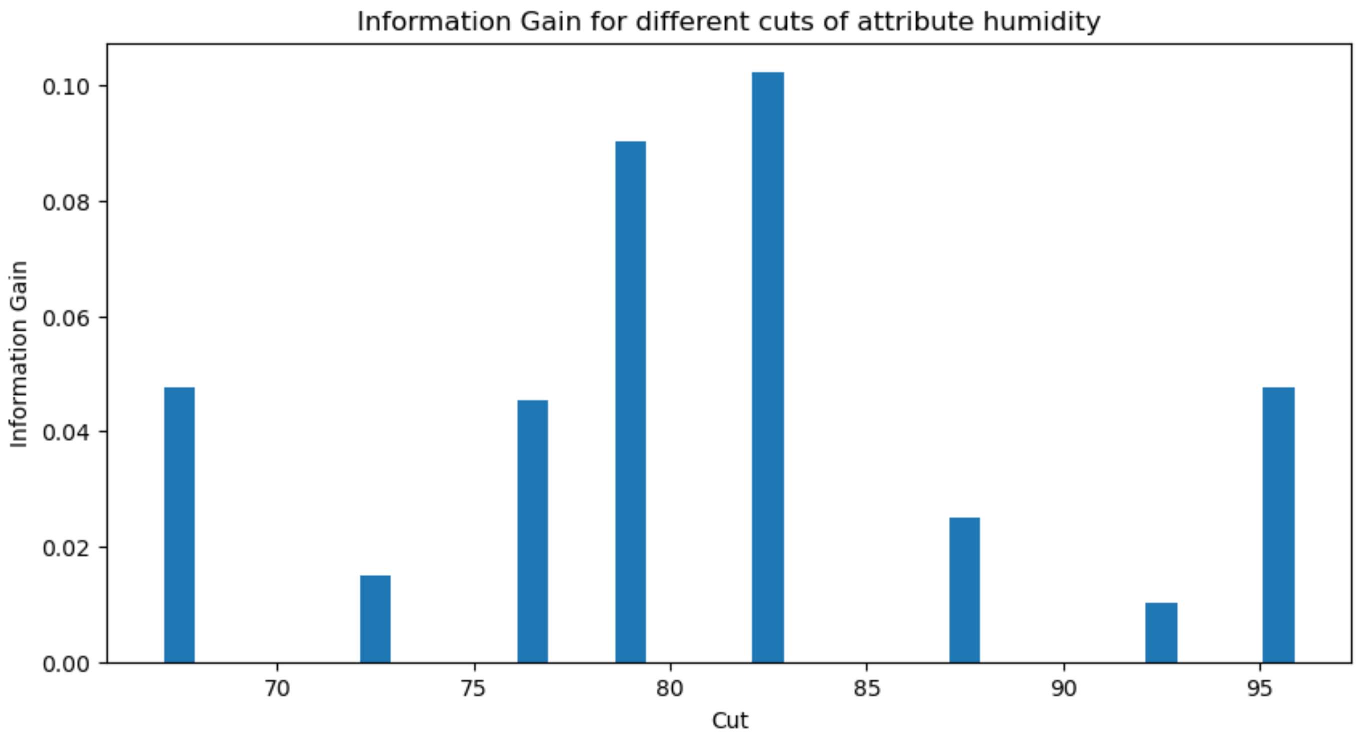
✓




```
Information gain forecast: 0.22600024438491684
Best cut forecast: cloudy
```




```
Information gain temperature: 0.1134008641811034
Best cut temperature: 28.85
```




```
Information gain humidity: 0.10224356360985076
Best cut humidity: 82.5
```

✓

d)

Answer: **weather forecast**

The best attribute to derive a decision is the **weather forecast**. The information gain is highest when a cut is made at the attribute value "**cloudy**".

A quick glance at the data shows why: On **all** cloudy days, the soccer game was played. The separation of the (sub) data sets between `soccer==True` and `soccer==False` is the "cleanest" (lowest entropy) for this attribute. ✓

$$\sum_{A17} = 5/5$$