XAI2_task3.py

```python
# ----------- Import libraries ----------------------------------------

import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from rulefit import RuleFit
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler


# ----------- Preprocessing data ----------------------------------------
data_task3= pd.read_csv('datasets/day.csv', sep=',')

data_task3['spring'] = np.where(data_task3['season'] == 2, 1, 0)
data_task3['summer'] = np.where(data_task3['season'] == 3, 1, 0)
data_task3['fall']   = np.where(data_task3['season'] == 4, 1, 0)

data_task3['misty'] = np.where(data_task3['weathersit'] == 2, 1, 0)
data_task3['rain']  = np.where(data_task3['weathersit'].isin([3, 4]), 1, 0)

data_task3['temp']     = data_task3['temp'].astype(float) * 47 - 8
data_task3['hum']      = data_task3['hum'].astype(float) * 100
data_task3['windspeed']= data_task3['windspeed'].astype(float) * 67

data_task3['dteday'] = pd.to_datetime(data_task3['dteday'])
data_task3['days_since_2011'] = (data_task3['dteday'] - pd.Timestamp('2011-01-01')).dt.days


# ----------- Finding f(x) = y ----------------------------------------
X = data_task3[['fall','spring','summer', 'workingday','holiday', 'misty','rain', 'temp','hum','windspeed',
'days_since_2011']]
y = data_task3['cnt']


# ----------- Training rulefit ----------------------------------------
gb = GradientBoostingRegressor(n_estimators=500, max_depth=2, learning_rate=0.01, random_state=13)


rf = RuleFit(                       # Implicitly gaussian, so no need to specify family="gaussian"
    tree_generator = gb,
    random_state   = 13
)

scaler = StandardScaler()               # Scaling values to take care for the different ranges of the
features, especially for the temp feature vs days_since_2011
X_scaled = scaler.fit_transform(X)

rf.fit(X_scaled, y.values, feature_names=X.columns.tolist())

# ----------- Finding the top 4 rules ----------------------------------------
rules = rf.get_rules()
rules_filtered = rules[
    (rules['importance'] != 0) &
    (rules['type'] != 'rule')
]

rules_top4 = (rules_filtered
        .sort_values(by='importance', ascending=False)
        .head(4)
        .reset_index(drop=True)
```

```
)

print("Top 4 rules:")
print(rules_top4[['rule', 'importance']])

# ----------- Plotting the top 4 rules ----------------------------------------
fig, ax = plt.subplots(figsize=(8,5))
bars = ax.bar(rules_top4['rule'], rules_top4['importance'])
ax.set_ylabel('Importance')
ax.set_title('Task 3 - Top 4 rules')
plt.tight_layout()

for bar in bars:
    h = bar.get_height()
    ax.annotate(f'{h:.1f}',
                xy=(bar.get_x() + bar.get_width()/2, h),
                xytext=(0, 3),
                textcoords="offset points",
                ha='center', va='bottom')

plt.show()
```