MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
Facultatea de Automatica si Calculatoare

Codul sursa
Climatul automatizat al unei sere

Student: **Lavinia-Maria ABRUDAN**

Disciplina: **Sisteme bazate pe Cunoaștere**

➢ Importul bibliotecilor necesare

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima
from scipy import signal
from scipy.ndimage import median_filter
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score
```

➢ Citirea si analiza datasetului

```
df = pd.read_csv("C:\\Users\\abrud\\OneDrive\\Desktop\\Anul 3\\Sisteme bazate pe cunoastere SBC\\archive 1\\IoTProcessed_Data.csv")
print(df.head())
# shape of the data
df.shape
#data information
df.info()
# describing the data
df.describe()
#column to list
df.columns.tolist()
# check for missing values:
df.isnull().sum()
#checking duplicate values
df.nunique()
```

➢ Pre-procesarea datelor

```
#Pentru coloana ce indica temperatura
quality_counts = df['tempreature'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
```

```
plt.title('Count Plot of Temperature')
plt.xlabel('Temperature')
plt.ylabel('Count')
plt.show()

#Pentru coloana ce indica umiditatea
quality_counts = df['humidity'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='blue')
plt.title('Count Plot of Humidity')
plt.xlabel('Humidity')
plt.ylabel('Count')
plt.show()

#Pentru coloana ce indica nivelul de apa
quality_counts = df['water_level'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='green')
plt.title('Count Plot of Level of water')
plt.xlabel('Water Level')
plt.ylabel('Count')
plt.show()

#Pentru fiecare nutrient incepand cu nutrientul N
quality_counts = df['N'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Nutrient N')
plt.xlabel('N')
plt.ylabel('Count')
plt.show()

#Pentru nutrientul P
quality_counts = df['P'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Nutrient P')
plt.xlabel('P')
plt.ylabel('Count')
plt.show()

#Pentru nutrientul K
quality_counts = df['K'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Nutrient K')
```

```
plt.xlabel('K')
plt.ylabel('Count') plt.show()
#Pentru actuatorul de oprire a ventilatorului
quality_counts = df['Fan_actuator_OFF'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel('Fan_actuator_OFF')
plt.ylabel('Count')
plt.show()

#Pentru actuatorul de pornire a ventilatorului
quality_counts = df['Fan_actuator_ON'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel('Fan_actuator_ON)
plt.ylabel('Count')
plt.show()

# Pentru oprirea pompei de apa
quality_counts = df['Watering_plant_pump_OFF'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel(Watering_plant_pump_OFF)
plt.ylabel('Count')
plt.show()

#Pentru pornirea pompei de apa
quality_counts = df['Watering_plant_pump_ON'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel(Watering_plant_pump_ON)
plt.ylabel('Count')
plt.show()

#Pentru actuatorul de pornire a pompei de apa
quality_counts = df[' Water_pump_actuator_ON'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel(Water_pump_actuator_ON)
plt.ylabel('Count')
plt.show()
```

#Pentru actuatorul de oprire a pompei de apa

```
quality_counts = df[' Water_pump_actuator_OFF'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='pink')
plt.title('Count Plot of Actuator')
plt.xlabel(Water_pump_actuator_OFF)
plt.ylabel('Count')
plt.show()
```

#Analize vizuale

```
sns.set_style("darkgrid")
numerical_columns = df.select_dtypes(include=["int64", "float64"]).columns
plt.figure(figsize=(14, len(numerical_columns)*3))
for idx, feature in enumerate(numerical_columns, 1):
    plt.subplot(len(numerical_columns), 2, idx)
    sns.histplot(df[feature], kde=True)
    plt.title(f"{feature} | Skewness: {round(df[feature].skew(), 2)}")
plt.tight_layout()
   plt.show()
```

#Analiza de corelatie

```
print(df.dtypes)
df_numeric = df.select_dtypes(include=["float64", "int64"])
df_numeric = df_numeric.dropna()
plt.figure(figsize=(15, 10))
sns.heatmap(df_numeric.corr(), annot=True, fmt='.2f', cmap='Pastel2', linewidths=2)
plt.title('Correlation Heatmap')
plt.show()
```

➢ Pre-procesarea setului de date

#Eliminarea zgomotului

```
signal = df['tempreature'].values[:1000]     #semnalul brut, reprezinta temperatura
time = np.arange(len(signal)) #acest set este analizat pentru primii 1000 pasi
temporali
plt.figure(figsize=(10, 6))
plt.plot(time, signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
```

Graficul semnalului original indica variatii neregulate cauzate de zgomot.

```
sampling_rate = 256
cutoff = 40
filtered_signals = {
        "Low-pass": low_pass_filter(signal, cutoff, sampling_rate),
        "High-pass": high_pass_filter(signal, 1, sampling_rate),
        "Band-pass": band_pass_filter(signal, 1, 40, sampling_rate),
        "Moving Mean": moving_mean_filter(signal),
        "Median": apply_median_filter(signal),
        "Adaptive": adaptive_filter(signal),
        "Wavelet": wavelet_denoising(signal)
}
plt.figure(figsize=(14, 10))
for i, (label, filtered_signal) in enumerate(filtered_signals.items(), start=1):
    plt.subplot(4, 2, i)
    plt.plot(time, filtered_signal, label=f'{label} Filter')
    plt.title(f'{label} Filtered Signal')
    plt.legend()
    plt.tight_layout()
plt.show()
```

#Reducerea dimensiunii

```
# Pregatirea datelor
numeric_data = df.select_dtypes(include=[np.number]).dropna()
#am eliminat valorile lipsa din coloanele numerice
# Aplicare PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(numeric_data)
# Vizualizarea rezultatelor PCA
plt.figure(figsize=(10, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c='blue', alpha=0.7)
plt.title("PCA Reduced Data")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

#Eliminearea tendintelor

```
signal = df['humidity']
time = np.arange(len(signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, signal, label='Original Signal')
plt.title("Original Signal")
```

```
plt.legend()
plt.show()
```

#Eliminarea tendintei folosind media mobila

```
window_size = 100
moving_avg = signal.rolling(window=window_size, center=True).mean()
detrended_moving_avg = signal - moving_avg
plt.figure(figsize=(10, 6))
plt.plot(time, signal, label='Original')
plt.plot(time, detrended_moving_avg, label='Moving Average Detrended')
plt.title("Moving Average Detrended Signal")
```

#Eliminarea tendintei folosind polinoame

```
polynomial_coeffs = np.polyfit(time, signal, 3)
polynomial_trend = np.polyval(polynomial_coeffs, time)
detrended_polynomial = signal - polynomial_trend
plt.figure(figsize=(10, 6))
plt.plot(time, signal, label='Original')
plt,plot(time, detrended_polynomial, label='Polynomial Detrended (Cubic)')
plt.title("Polynomial Detrended Signal")
plt.legend()
plt.show()
```

#Interpolarea esantioanelor lipsa

```
water_signal = df['water_level']
time = np.arange(len(water_signal))
np.random.seed(0)
signal_missing = water_signal.copy()
missing_indices = np.random.choice(len(water_signal), size=2000, replace=False)
signal_missing[missing_indices] = np.nan
plt.figure(figsize=(10, 6))
plt.plot(time, signal_missing, label='Original Signal with Missing Values', color='pink')
plt.title("Original Signal with Missing Values")
plt.legend()
plt.show()
```

#Eliminarea valorilor aberante pentru coloana ce indica nivelul apei

```
water_signal = df['water_level']
time = np.arange(len(water_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, water_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
```

```
water_signal_reshaped = water_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(water_signal_reshaped)
signal_no_outliers_dbscan = water_signal[dbscan_labels != -1]
plt.figure(figsize=(10, 6))
plt.plot(time, water_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels!=-1],signal_no_outliers_dbscan,label="DBSCAN Filtered",
color='grey')
plt.title("DBSCAN Outlier Removal")
plt.legend()
plt.show()
```

#Eliminarea valorilor aberante pentru coloana ce indica umiditatea

```
humidity_signal = df['humidity']
time = np.arange(len(humidity_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, humidity_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
humidity_signal_reshaped = humidity_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(humidity_signal_reshaped)
signal_no_outliers_dbscan = humidity_signal[dbscan_labels != -1]
plt.figure(figsize=(10, 6))
plt.plot(time, humidity_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels!=-1],signal_no_outliers_dbscan,label="DBSCAN Filtered",
  color='grey')
plt.title("DBSCAN Outlier Removal")
plt.legend()
plt.show()
```

#Eliminarea valorilor aberante pentru coloana ce indica temperatura

```
temperature_signal = df['tempreature']
time = np.arange(len(temperature_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, temperature_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
temperature_signal_reshaped = temperature_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(temperature_signal_reshaped)
signal_no_outliers_dbscan = temperature_signal[dbscan_labels != -1]
```

```
plt.figure(figsize=(10, 6))
plt.plot(time, temperature_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels != -1], signal_no_outliers_dbscan, label="DBSCAN
Filtered", color='grey')
plt.title("DBSCAN Outlier Removal")
plt.legend()
plt.show()
```

#Eliminarea valorilor aberante pentru coloana fiecarui nutrient

```
N_signal = df['N']
time = np.arange(len(N_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, N_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
N_signal_reshaped = N_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(N_signal_reshaped)
signal_no_outliers_dbscan = N_signal[dbscan_labels != -1]
plt.figure(figsize=(10, 6))
plt.plot(time, N_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels != -1], signal_no_outliers_dbscan, label="DBSCAN
Filtered", color='grey')
plt.title("DBSCAN Outlier Removal")
plt.legend()
plt.show()

P_signal = df['P']
time = np.arange(len(P_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, P_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
P_signal_reshaped = P_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(P_signal_reshaped)
signal_no_outliers_dbscan = P_signal[dbscan_labels != -1]
plt.figure(figsize=(10, 6))
plt.plot(time, P_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels != -1], signal_no_outliers_dbscan, label="DBSCAN
Filtered", color='grey')
plt.title("DBSCAN Outlier Removal")
```

```
plt.legend()
plt.show()

K_signal = df['K']
time = np.arange(len(K_signal))
# Plot original signal
plt.figure(figsize=(10, 6))
plt.plot(time, K_signal, label='Original Signal')
plt.title("Original Signal")
plt.legend()
plt.show()
K_signal_reshaped = K_signal.values.reshape(-1, 1)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(K_signal_reshaped)
signal_no_outliers_dbscan = K_signal[dbscan_labels != -1]

plt.figure(figsize=(10, 6))
plt.plot(time, K_signal, label="Original Signal", color='blue', alpha=0.5)
plt.plot(time[dbscan_labels != -1], signal_no_outliers_dbscan, label="DBSCAN
Filtered", color='grey')
plt.title("DBSCAN Outlier Removal")
plt.legend()
plt.show()
```

➢ Construirea modelelor

#Impartirea setului de date in date de testare si date de validare

```
X=df[['tempreature','humidity','water_level','N','P','K']]
y=df[['Fan_actuator_OFF','Fan_actuator_ON','Watering_plant_pump_ON','Water_pu
mp_actuator_ON']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
models = {}

for column in y_train.columns:
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train[column])
    models[column] = model

regressor = LinearRegression()
regressor.fit(X_train, y_train['Water_pump_actuator_ON'])
for column, model in models.items():
    predictions = model.predict(X_test)
    print(f"Evaluare pentru {column}:")
    print("Acuratete:", accuracy_score(y_test[column], predictions))
```

```
    print("Raport clasificare:\n", classification_report(y_test[column], predictions))
    print("Matrice de confuzie:\n", confusion_matrix(y_test[column], predictions))
    print("-" * 50)


for column, model in models.items():
    predictions = model.predict(X_test)
    plt.figure(figsize=(6, 4))
    sns.heatmap(confusion_matrix(y_test[column], predictions), annot=True, fmt='d',
cmap='Blues')
    plt.title(f"Matrice de confuzie pentru {column}")
    plt.xlabel("Predictie")
    plt.ylabel("Valoare reala")
    plt.show()
```

#Random Forest Classifier

```
# Initializez clasificatorul pentru Fan_actuator_ON
model_fan = RandomForestClassifier(n_estimators=100, random_state=42)
model_fan.fit(X_train, y_train['Fan_actuator_ON'])

y_pred_fan = model_fan.predict(X_test)
print("Evaluare pentru Fan_actuator_ON:")
print("Acuratete:", accuracy_score(y_test['Fan_actuator_ON'], y_pred_fan))
print("Raport clasificare:\n", classification_report(y_test['Fan_actuator_ON'],
y_pred_fan))
print("Matrice de confuzie:\n", confusion_matrix(y_test['Fan_actuator_ON'],
  y_pred_fan))
print("-" * 50)

model_pump = RandomForestClassifier(n_estimators=100, random_state=42)
model_pump.fit(X_train, y_train['Watering_plant_pump_ON'])
y_pred_pump = model_pump.predict(X_test)
print("Evaluare pentru Watering_plant_pump_ON:")
print("Acuratete:", accuracy_score(y_test['Watering_plant_pump_ON'],
y_pred_pump))
print("Raport clasificare:\n", classification_report(y_test['Watering_plant_pump_ON' ]
, y_pred_pump))
print("Matrice de confuzie:\n", confusion_matrix(y_test['Watering_plant_pump_ON'],
y_pred_pump))
print("-" * 50)


model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Predictie
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification = classification_report(y_test, y_pred, target_names=y)

def plot_predictions(y_test, y_pred, title):
    plt.figure(figsize=(12, 6))
    plt.plot(y_test[:100].values, label="Actual", marker='o', linestyle='dashed',
alpha=0.7,color='blue')
    plt.plot(y_pred[:100], label="Predicted", marker='x', linestyle='dotted',
alpha=0.7,color='red')
    plt.xlabel("Sample Index")
    plt.ylabel("Actuator State (ON=1, OFF=0)")
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()

for actuator in actuator_cols:
    y_pred = models[actuator].predict(X_test)
    plot_predictions(y_test[actuator], y_pred, f"Semnal actual vs predictie: {actuator}")


# Rezultate
print(f"Acuratetea modelului: {accuracy:.4f}")
print("Raport de clasificare:")
print(classification)

# Importanta caracteristicilor
importance_df = pd.DataFrame({'Feature': X, 'Importance':
model.feature_importances_})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

for actuator in actuator_cols:
    importances = models[actuator].feature_importances_
    indices = np.argsort(importances)[::-1]
    plt.figure(figsize=(10, 8))
    plt.title(f"Caracteristici importante pentru {actuator}")
    plt.bar(range(X.shape[1]), importances[indices], align="center")
    plt.xticks(range(X.shape[1]), X.columns[indices], rotation=45)
    plt.xlabel("Caracteristici")
    plt.ylabel("Importanta")
    plt.show()
```

```
# Vizualizare importanta caracteristici
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
plt.xlabel("Importanta")
plt.ylabel("Caracteristica")
plt.title("Importanta caracteristicilor in modelul Random Forest Classifier")
plt.show()
```

➢ Implementarea

```
def control_actuators(df):
    for index, row in df.iterrows():

        # Conditia pentru pompa de apa
          if row['humidity']<40:
              df.at[index, 'Watering_plant_pump_ON'] = 1
              df.at[index, 'Watering_plant_pump_OFF'] = 0
              print(f"[Row {index}] Pompa de apa PORNITA pentru umiditate scazuta.")
          elif row['humidity'] > 70:
              df.at[index, 'Watering_plant_pump_ON'] = 0
              df.at[index, 'Watering_plant_pump_OFF'] = 1
              print(f"[Row {index}] Pompa de apa OPRITA pentru umiditate optima.")

         # Conditia pentru ventilator
          if row['tempreature']>35:
              df.at[index, 'Fan_actuator_ON'] = 1
              df.at[index, 'Fan_actuator_OFF'] = 0
              print(f"[Row {index}] Ventilator PORNIT pentru temperatura foarte
    ridicata.")
          elif row['tempreature']<25:
                  df.at[index, 'Fan_actuator_ON'] = 0
                  df.at[index, 'Fan_actuator_OFF'] = 1
                  print(f"[Row {index}] Ventilator OPRIT pentru ca temperatura este
    scazuta.")

        # Conditia pentru nivelul apei
            if row['water_level'] < 30:
                  df.at[index, 'Water_pump_actuator_ON'] = 1
                  df.at[index, 'Water_pump_actuator_OFF'] = 0
                  print(f"[Row {index}] Pompa de apa PORNITA pentru nivel scazut al
    apei.")
            elif row['water_level'] > 70:
                  df.at[index, 'Water_pump_actuator_ON'] = 0
                  df.at[index, 'Water_pump_actuator_OFF'] = 1
                  print(f"[Row {index}] Pompa de apa OPRITA pentru nivel optim al
    apei.")
```

```
        # Conditia pentru nutrienti
        if 'Soil_Nutrients' in row and row['Soil_Nutrients'] < 50:
            print(f"[Row {index}] Solul are nivel scazut de nutrienti.")
        elif 'Soil_Nutrients' in row and row['Soil_Nutrients'] > 200:
            print(f"[Row {index}] Solul are nivel ridicat de nutrienti.")

return df
```