

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# =====
# Scikit-learn Models
# =====
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor
)
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.multioutput import MultiOutputRegressor
# =====
# Scikit-learn Utilities
# =====
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score
)
# =====
# XGBoost
# =====
from xgboost import XGBRegressor
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/IoTpond1.csv")
df.head()

```

```
df.isnull().sum()
```

```

df.drop(['created_at'], axis=1, inplace=True)
plt.figure(figsize=(8, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

```

```

plt.figure(figsize=(8, 6))
sns.histplot(df['Ammonia(g/ml)'], bins=15, kde=True, color='blue')
plt.title('Ammonia Distribution')
plt.xlabel('Ammonia')
plt.ylabel('Frequency')
plt.show()

```

```

def detect_outliers_iqr(df, columns, k=1.5):
    outliers = pd.DataFrame(False, index=df.index, columns=columns)
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - k * IQR
        upper = Q3 + k * IQR
        outliers[col] = (df[col] < lower) | (df[col] > upper)
    return outliers
# Example
outliers = detect_outliers_iqr(df, ["Ammonia(g/ml)"])
df[outliers.any(axis=1)]

```

```

df.drop(['Ammonia(g/ml)'], axis=1, inplace=True)
df.fillna(df.mean(), inplace=True)
df.isnull().sum()

```

```

X= df.drop(['Fish_Length(cm)', 'Fish_Weight(g)'], axis=1)
y = df[['Fish_Length(cm)', 'Fish_Weight(g)']]
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

```

rfr = RandomForestRegressor(n_estimators=200, random_state=42)
multi_rfr = MultiOutputRegressor(rfr)
# Train
multi_rfr.fit(X_train, y_train)
# Predict
y_pred = multi_rfr.predict(X_test)
# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display results
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")

```

```

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (RFR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()

```

```

dtr = DecisionTreeRegressor(random_state=42)
multi_dtr = MultiOutputRegressor(dtr)
# Train model
multi_dtr.fit(X_train, y_train)
# Predict
y_pred = multi_dtr.predict(X_test)
# Evaluate performance
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")
# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (DTR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()

```

```

from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd

# Load the dataset and perform preprocessing steps for df
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/IoTpond1.csv")
df.drop(['created_at'], axis=1, inplace=True)
df.drop(['Ammonia(g/ml)'], axis=1, inplace=True)
df.fillna(df.mean(), inplace=True)

```

```
# Re-defining X, y, X_train, y_train, X_test, y_test to ensure they are available
X = df.drop(['Fish_Length(cm)', 'Fish_Weight(g)'], axis=1)
y = df[['Fish_Length(cm)', 'Fish_Weight(g)']]
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svr = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
multi_svr = MultiOutputRegressor(svr)
# Train model
multi_svr.fit(X_train, y_train)
# Predict
y_pred = multi_svr.predict(X_test)
# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (SVR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

```
xgb = XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="reg:squarederror",
    random_state=42,
    n_jobs=-1
```

```
model = MultiOutputRegressor(xgb)
# =====
# Train
# =====
model.fit(X_train, y_train)
# =====
# Predict
# =====
y_pred = model.predict(X_test)
# =====
# Evaluation
# =====
mae = mean_absolute_error(y_test, y_pred, multioutput="raw_values")
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput="raw_values"))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y_test.shape[1])]
# =====
# Display Metrics
# =====
for i, col in enumerate(y_test.columns):
    print(f"\nTarget: {col}")
    print(f"MAE : {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R² : {r2[i]:.3f}")
```

```
fig, axes = plt.subplots(1, y_test.shape[1], figsize=(6 *
y_test.shape[1], 5))
if y_test.shape[1] == 1:
    axes = [axes]
for i, col in enumerate(y_test.columns):
    axes[i].scatter(
        y_test.iloc[:, i],
        y_pred[:, i],
        alpha=0.7,
        edgecolors='k'
    )
    axes[i].plot(
        [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
        [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
        'r--', lw=2
    )
    axes[i].set_xlabel("Actual Values")
    axes[i].set_ylabel("Predicted Values")
    axes[i].set_title(f"{col} - Actual vs Predicted (XGBoost)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```