Combined PID Gain Classifier + PID vs Fuzzy Comparison_Final

December 15, 2025

[1]:
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score



from sklearn.svm import SVC

import warnings
warnings.filterwarnings("ignore")
```

[2]:
```python
# 1) Load dataset

DATA_PATH = r"I:\Self Study\python study\A Practical Industrial ML Applications␣
↪for Smart Manufacturing Temperature Regulation\Smart Manufacturing␣
↪Temperature Regulation Dataset.csv"
BASE_DIR = os.path.dirname(DATA_PATH)
OUT_DIR = os.path.join(BASE_DIR, "outputs")
os.makedirs(OUT_DIR, exist_ok=True)

def ts():
```

```
    return time.strftime("%Y%m%d_%H%M%S")

print(f"Data path: {DATA_PATH}")
print(f"Outputs will be saved to: {OUT_DIR}")
```

```
Data path: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\Smart Manufacturing Temperature
Regulation Dataset.csv
Outputs will be saved to: I:\Self Study\python study\A Practical Industrial ML
Applications for Smart Manufacturing Temperature Regulation\outputs
```

[3]:
```
#  Load data
df = pd.read_csv(DATA_PATH)
print(f"Loaded dataset with shape: {df.shape}")
```

```
Loaded dataset with shape: (1000, 15)
```

[4]:
```
# 2) Dynamic Feature Engineering

df["Temperature Error_lag1"] = df["Temperature Error (°C)"].shift(1)
df["Temperature_delta"] = df["Current Temperature (°C)"].diff()
df["PID_Output_rollmean3"] = df["PID Control Output (%)"].rolling(3,␣
↪min_periods=1).mean()
df["Ambient_Temp_delta"] = df["Ambient Temperature (°C)"].diff()

df = df.dropna().reset_index(drop=True)
```

[5]:
```
# 3) Create Classification Targets (PID & Fuzzy)
# Create Kp_class, Ki_class, Kd_class (for classifiers)

df["Kp_class"] = pd.qcut(df["PID Kp"], q=3, labels=["Low", "Medium", "High"])
df["Ki_class"] = pd.qcut(df["PID Ki"], q=3, labels=["Low", "Medium", "High"])
df["Kd_class"] = pd.qcut(df["PID Kd"], q=3, labels=["Low", "Medium", "High"])

print("\nKp_class distribution:")
print(df["Kp_class"].value_counts())

print("\nKi_class distribution:")
print(df["Ki_class"].value_counts())

print("\nKd_class distribution:")
print(df["Kd_class"].value_counts())

df["PID_output_class"] = pd.qcut(
    df["PID Control Output (%)"], q=3, labels=["Low","Medium","High"]
)

df["Fuzzy_PID_output_class"] = pd.qcut(
```

```
    df["Fuzzy PID Control Output (%)"], q=3, labels=["Low","Medium","High"]
)
```

```
Kp_class distribution:
Kp_class
Low       333
Medium    333
High      333
Name: count, dtype: int64

Ki_class distribution:
Ki_class
Low       333
Medium    333
High      333
Name: count, dtype: int64

Kd_class distribution:
Kd_class
Low       333
Medium    333
High      333
Name: count, dtype: int64
```

[11]:
```
# 4) Defining the input feature set

features = [
    "Current Temperature (°C)",
    "Setpoint Temperature (°C)",
    "Temperature Error (°C)",
    "Ambient Temperature (°C)",
    "Humidity (%)",
    "PID Control Output (%)",
    "Fuzzy PID Control Output (%)",
    "Fuzzy Rule Base Parameters",
    "Temperature Error_lag1",
    "Temperature_delta",
    "PID_Output_rollmean3",
    "Ambient_Temp_delta"
]
X = df[features]
y = df["Kp_class"]

# Define the classification targets
classification_targets = {
    "Kp_class": df["Kp_class"],
```

```
    "Ki_class": df["Ki_class"],
    "Kd_class": df["Kd_class"],
    "PID_output_class": pd.qcut(df["PID Control Output (%)"], q=3,␣
↪labels=["Low", "Medium", "High"]),
    "Fuzzy_PID_output_class": pd.qcut(df["Fuzzy PID Control Output (%)"], q=3,␣
↪labels=["Low", "Medium", "High"])
}

target_cols = ["PID Kp", "PID Ki", "PID Kd"]
```

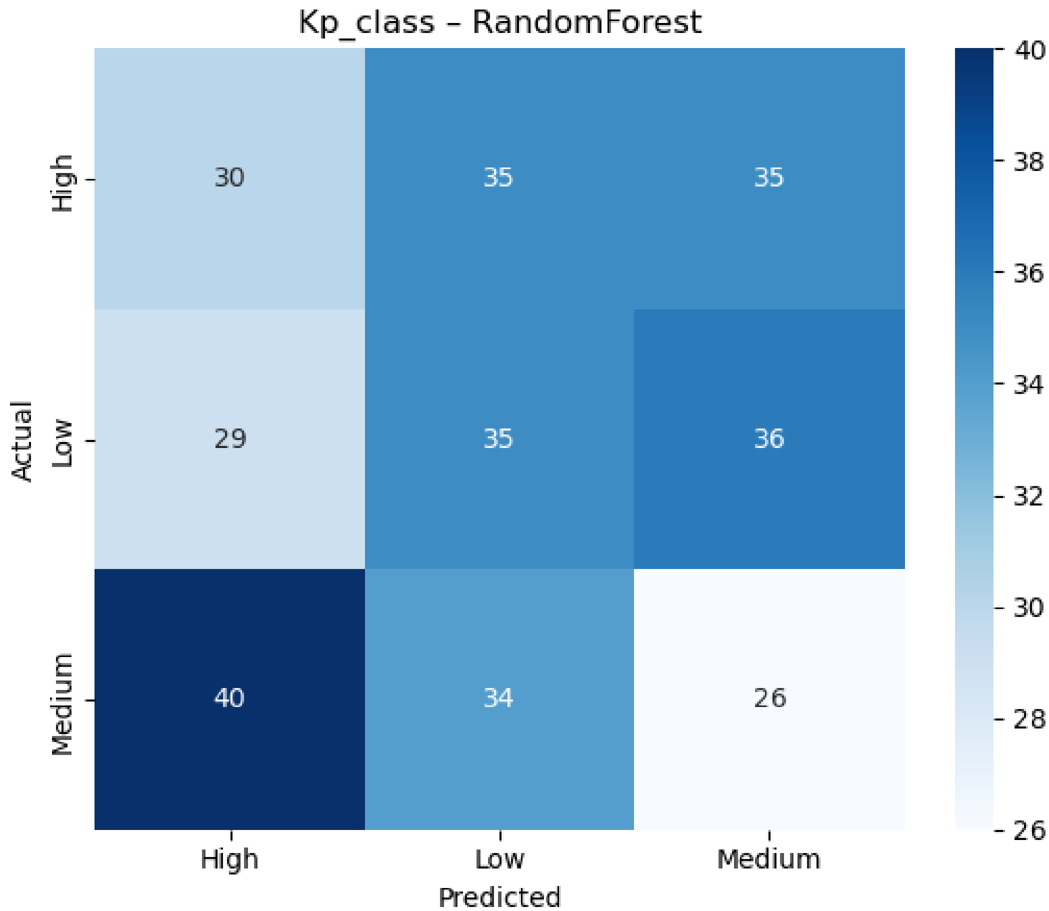
[13]:
```
# 5) Define ML Models

models = {
    "RandomForest": RandomForestClassifier(n_estimators=300, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "LogisticRegression": LogisticRegression(max_iter=500),
    "DecisionTree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel="rbf", probability=True)
}
```

[15]:
```
# 6) Define Training & Evaluation Function

def train_and_evaluate(models, X_train_s, X_test_s, y_train, y_test, OUT_DIR,␣
↪target_name):

    class_names = sorted(y_test.unique())

    for model_name, model in models.items():
        print(f"\nTraining {model_name} → {target_name}")

        model.fit(X_train_s, y_train)
        preds = model.predict(X_test_s)

        print(classification_report(y_test, preds))

        cm = confusion_matrix(y_test, preds)

        plt.figure(figsize=(6,5))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=class_names,
                    yticklabels=class_names)
        plt.title(f"{target_name} - {model_name}")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.tight_layout()

        save_path = os.path.join(OUT_DIR, f"CM_{target_name}_{model_name}.png")
```

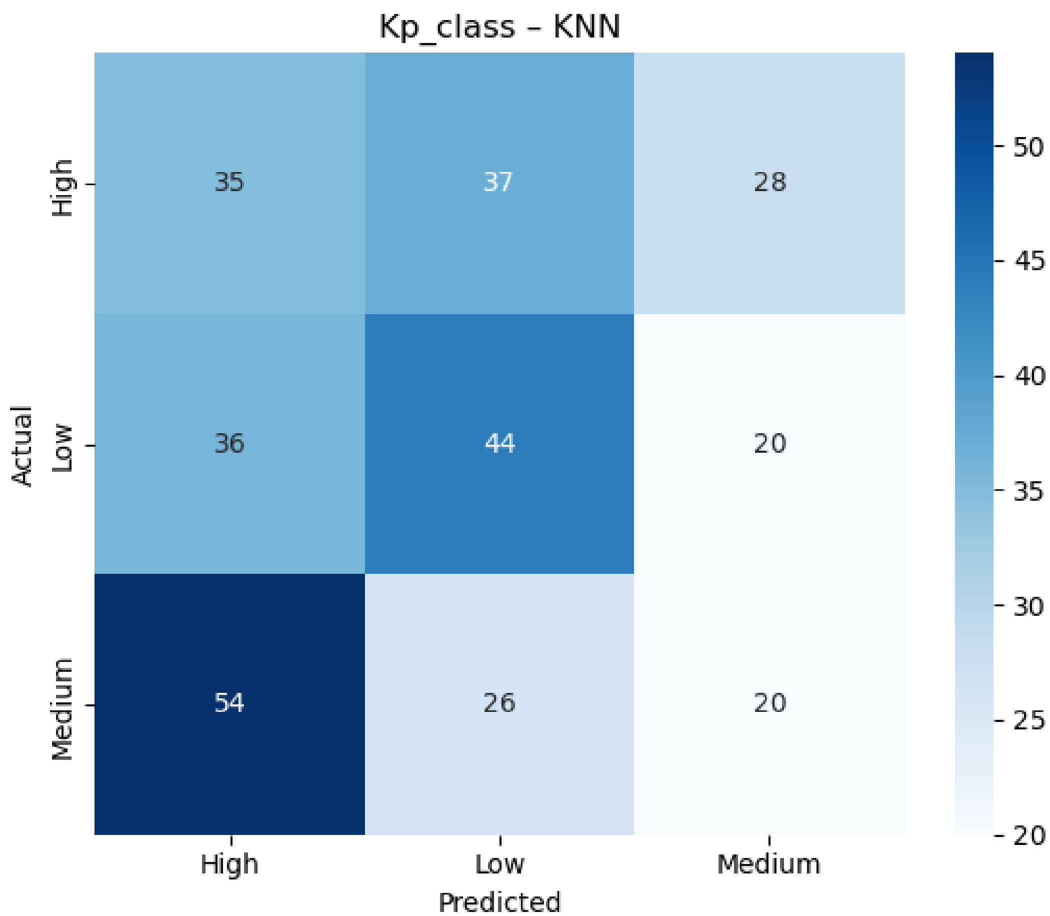
```
        plt.savefig(save_path, dpi=300)
        plt.show()

        print("Saved:", save_path)
```

[17]:
```
# 7) Train, Validate & Generate Confusion Matrices

for target_name, y in classification_targets.items():

    print("\n" + "="*70)
    print(f"PROCESSING TARGET: {target_name}")
    print("="*70)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, stratify=y, random_state=42
    )

    scaler = StandardScaler()
    X_train_s = scaler.fit_transform(X_train)
    X_test_s = scaler.transform(X_test)

    train_and_evaluate(
        models=models,
        X_train_s=X_train_s,
        X_test_s=X_test_s,
        y_train=y_train,
        y_test=y_test,
        OUT_DIR=OUT_DIR,
        target_name=target_name
    )
```

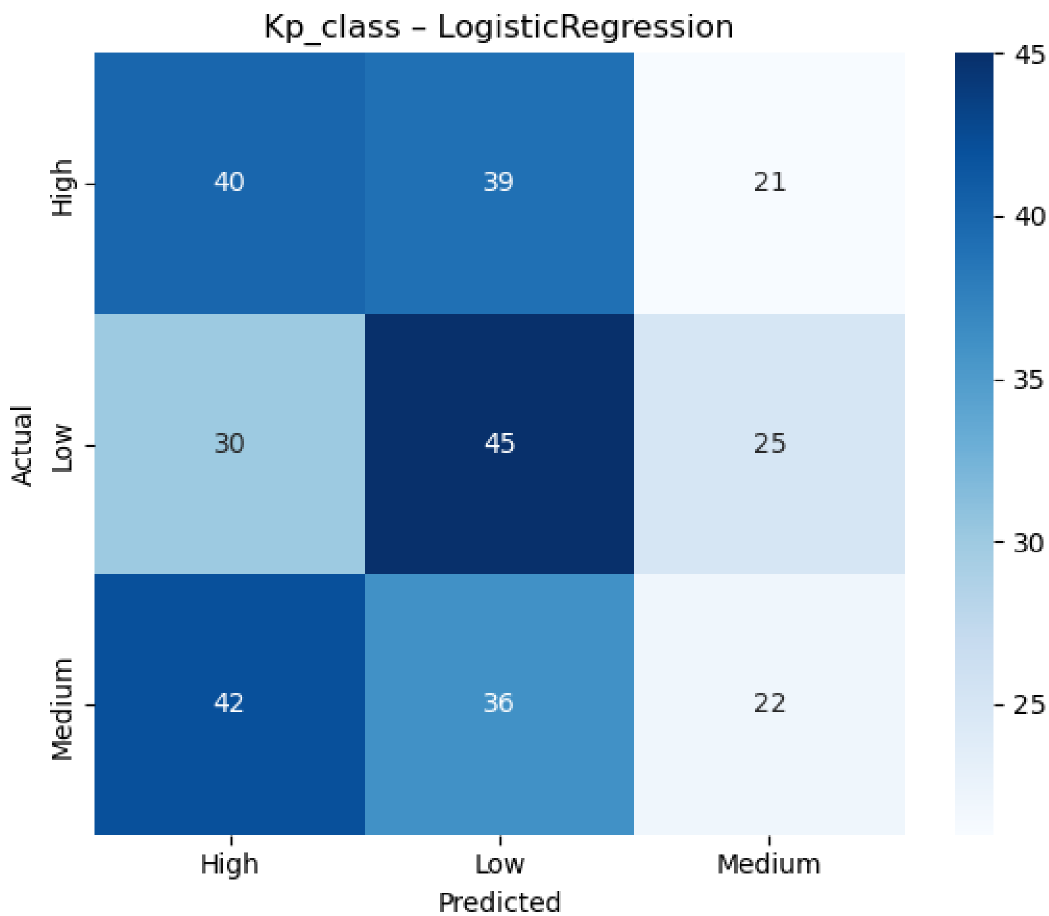
```
======================================================================
PROCESSING TARGET: Kp_class
======================================================================

Training RandomForest → Kp_class
              precision    recall  f1-score   support

        High       0.30      0.30      0.30       100
         Low       0.34      0.35      0.34       100
      Medium       0.27      0.26      0.26       100

    accuracy                           0.30       300
   macro avg       0.30      0.30      0.30       300
weighted avg       0.30      0.30      0.30       300
```



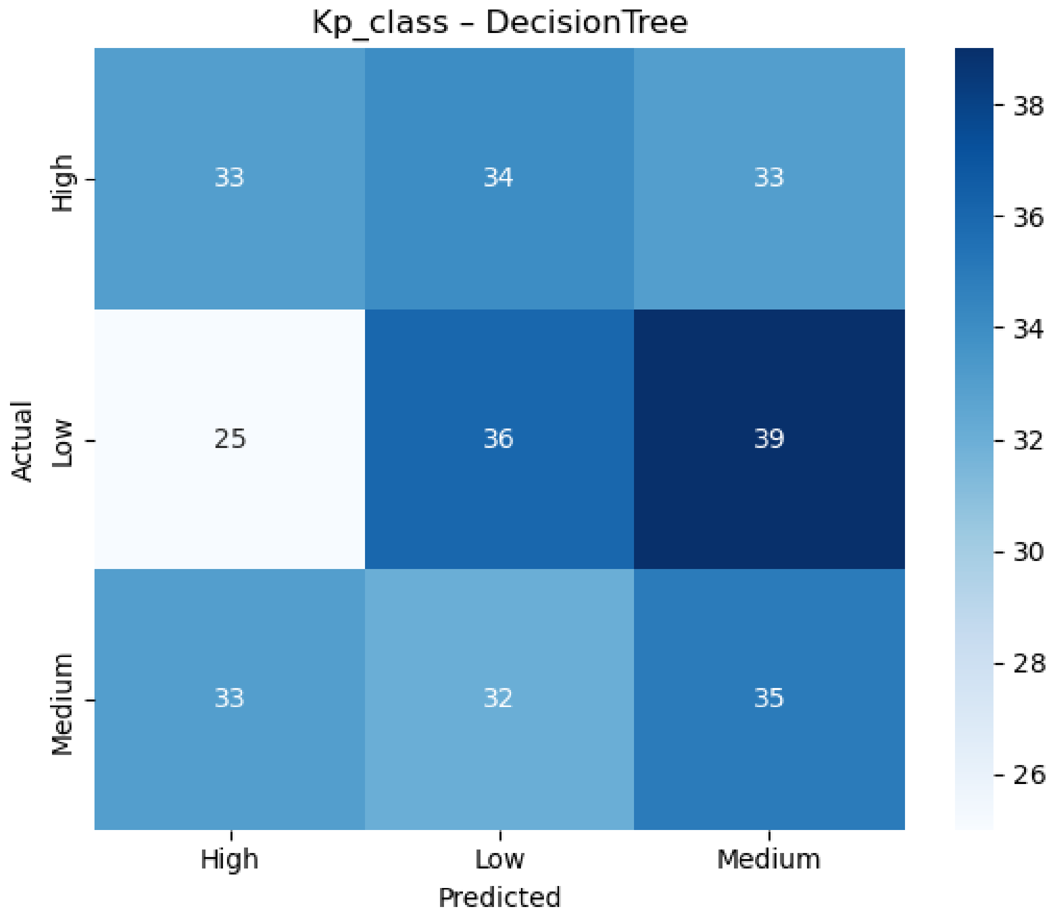

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kp_class_RandomForest.png

Training KNN → Kp_class
              precision    recall  f1-score   support

        High       0.28      0.35      0.31       100
         Low       0.41      0.44      0.43       100
      Medium       0.29      0.20      0.24       100

    accuracy                           0.33       300
   macro avg       0.33      0.33      0.32       300
weighted avg       0.33      0.33      0.32       300
```



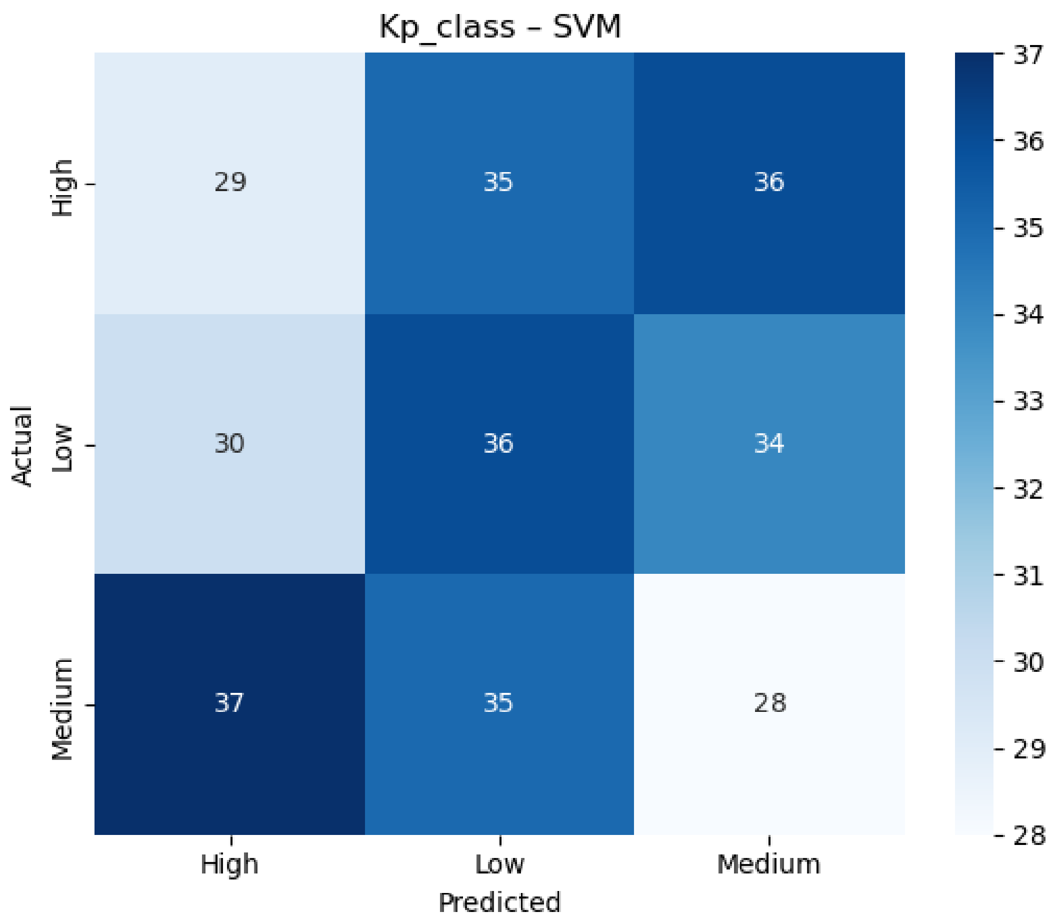

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kp_class_KNN.png

Training LogisticRegression → Kp_class
              precision    recall  f1-score   support

        High       0.36      0.40      0.38       100
         Low       0.38      0.45      0.41       100
      Medium       0.32      0.22      0.26       100

    accuracy                           0.36       300
   macro avg       0.35      0.36      0.35       300
weighted avg       0.35      0.36      0.35       300
```



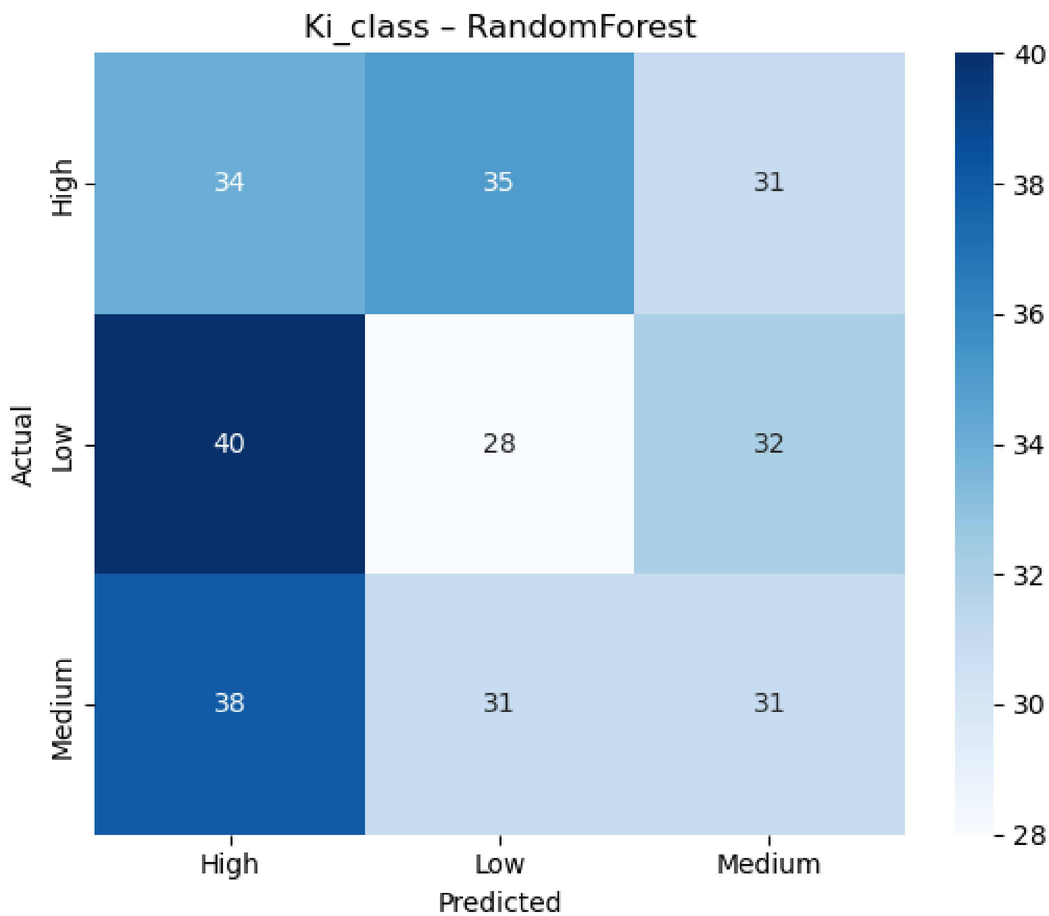

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Kp_class_LogisticRegression.png

Training DecisionTree → Kp_class
              precision    recall  f1-score   support

        High       0.36      0.33      0.35       100
         Low       0.35      0.36      0.36       100
      Medium       0.33      0.35      0.34       100

    accuracy                           0.35       300
   macro avg       0.35      0.35      0.35       300
weighted avg       0.35      0.35      0.35       300
```




```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kp_class_DecisionTree.png

Training SVM → Kp_class
              precision    recall  f1-score   support

        High       0.30      0.29      0.30       100
         Low       0.34      0.36      0.35       100
      Medium       0.29      0.28      0.28       100

    accuracy                           0.31       300
   macro avg       0.31      0.31      0.31       300
weighted avg       0.31      0.31      0.31       300
```



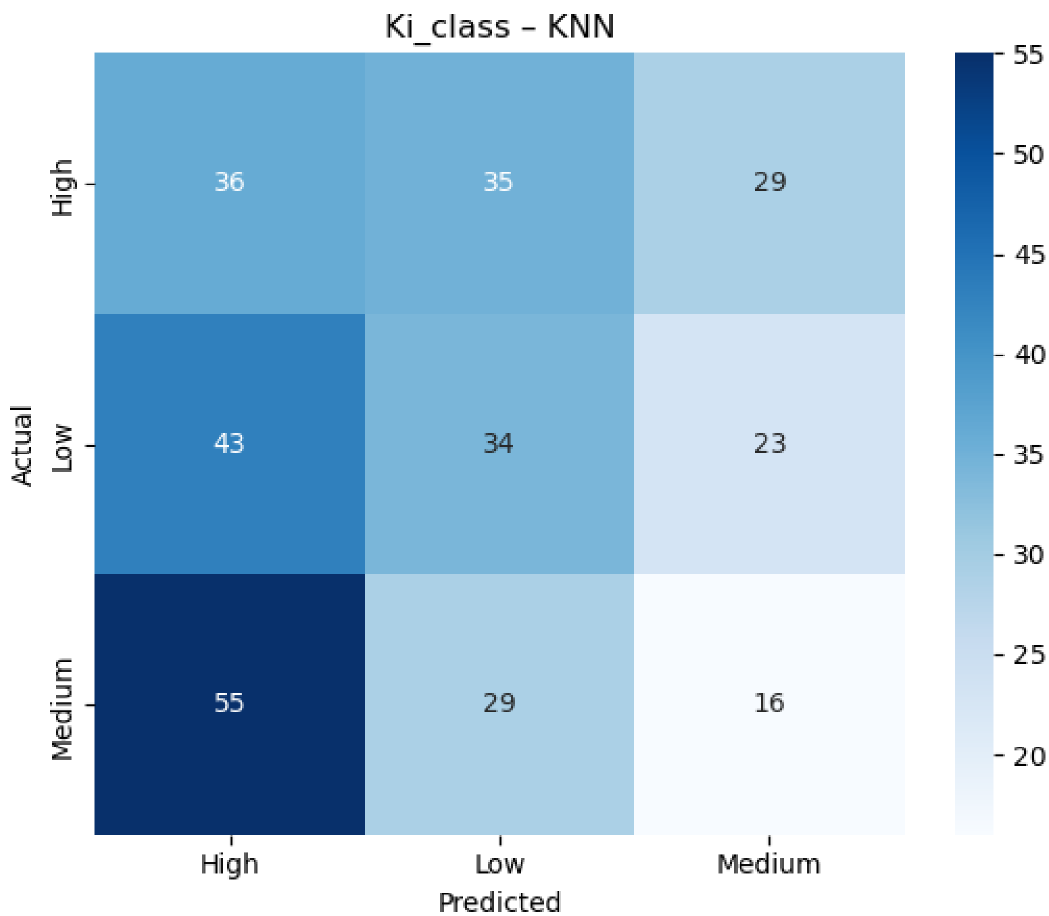

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kp_class_SVM.png

======================================================================
PROCESSING TARGET: Ki_class
======================================================================

Training RandomForest → Ki_class
              precision    recall  f1-score   support

        High       0.30      0.34      0.32       100
         Low       0.30      0.28      0.29       100
      Medium       0.33      0.31      0.32       100

    accuracy                           0.31       300
   macro avg       0.31      0.31      0.31       300
weighted avg       0.31      0.31      0.31       300
```



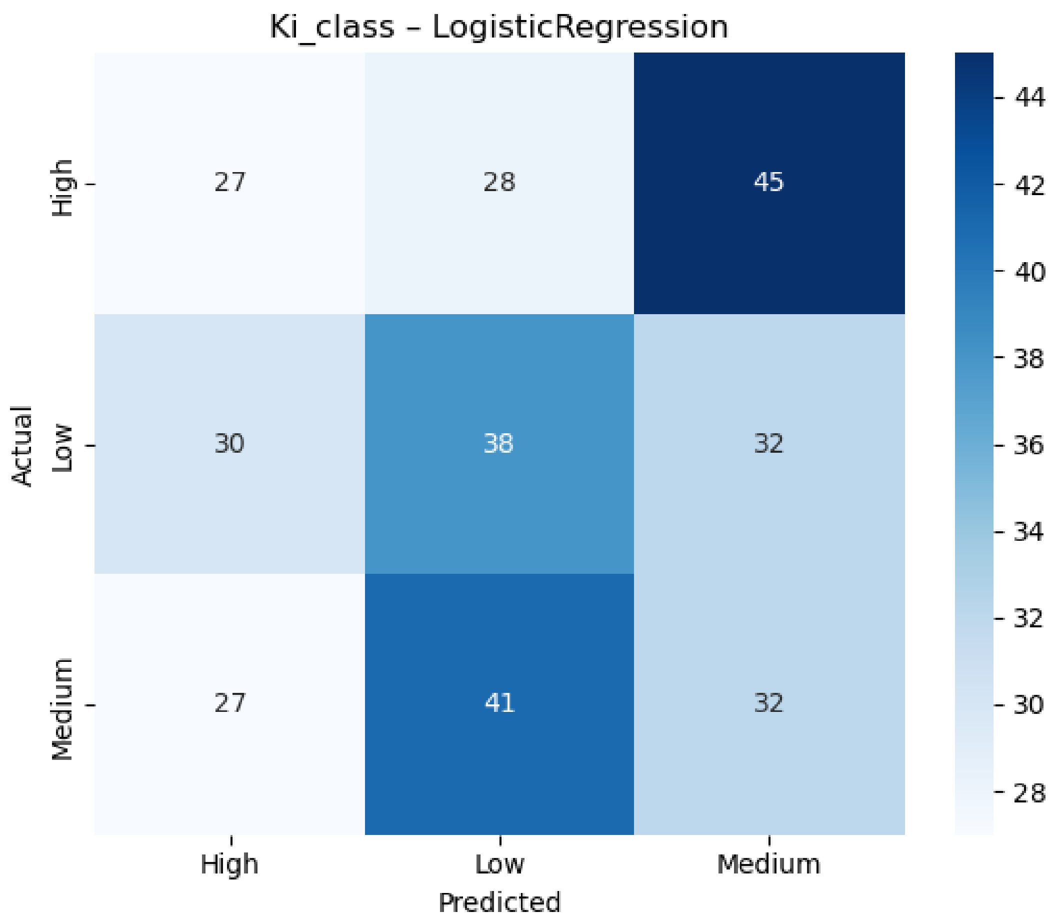
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\CM_Ki_class_RandomForest.png

Training KNN → Ki_class

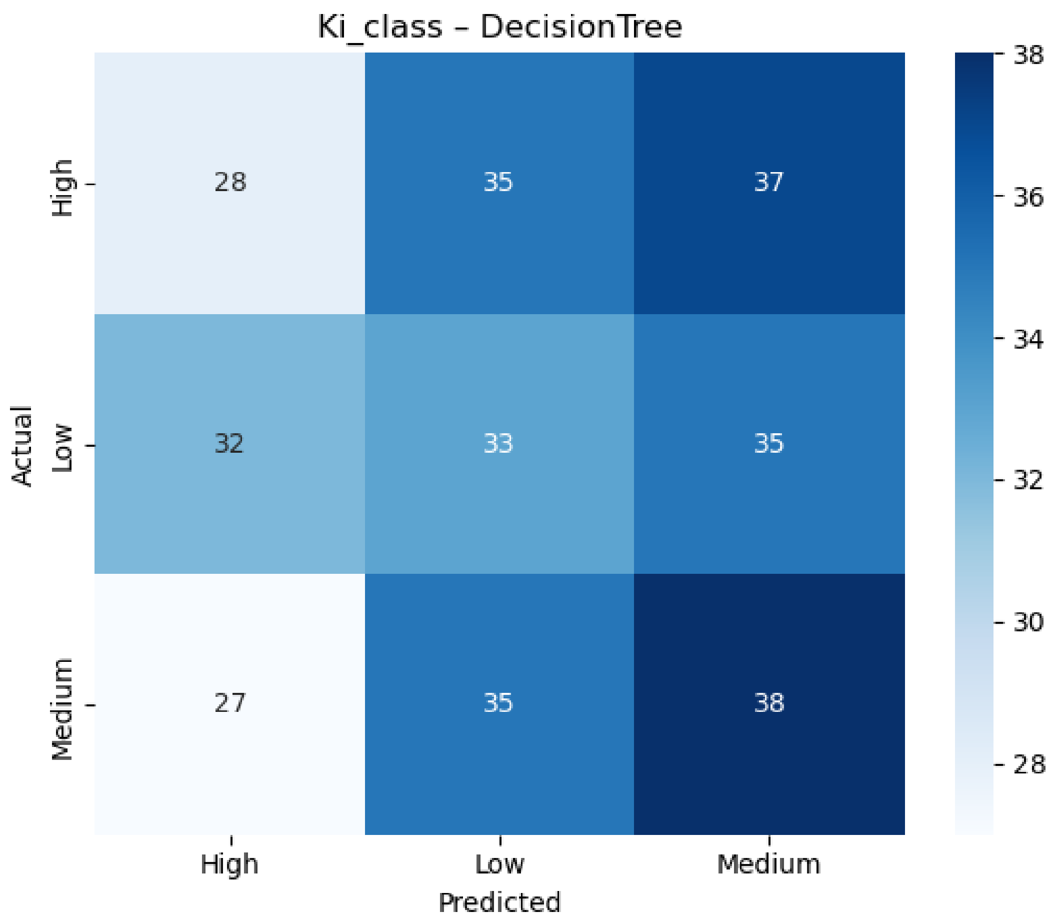
	precision	recall	f1-score	support
High	0.27	0.36	0.31	100
Low	0.35	0.34	0.34	100
Medium	0.24	0.16	0.19	100
accuracy			0.29	300
macro avg	0.28	0.29	0.28	300
weighted avg	0.28	0.29	0.28	300



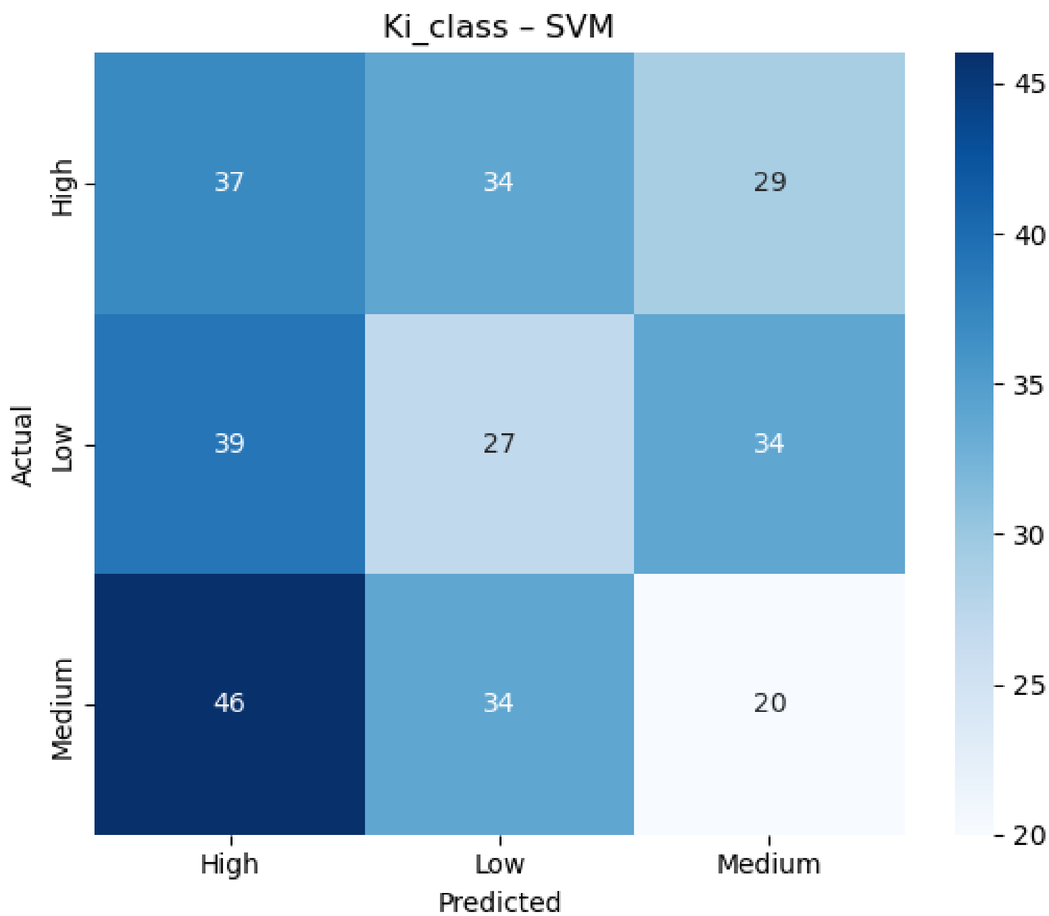

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Ki_class_KNN.png

Training LogisticRegression → Ki_class
              precision    recall  f1-score   support

        High       0.32      0.27      0.29       100
         Low       0.36      0.38      0.37       100
      Medium       0.29      0.32      0.31       100

    accuracy                           0.32       300
   macro avg       0.32      0.32      0.32       300
weighted avg       0.32      0.32      0.32       300
```



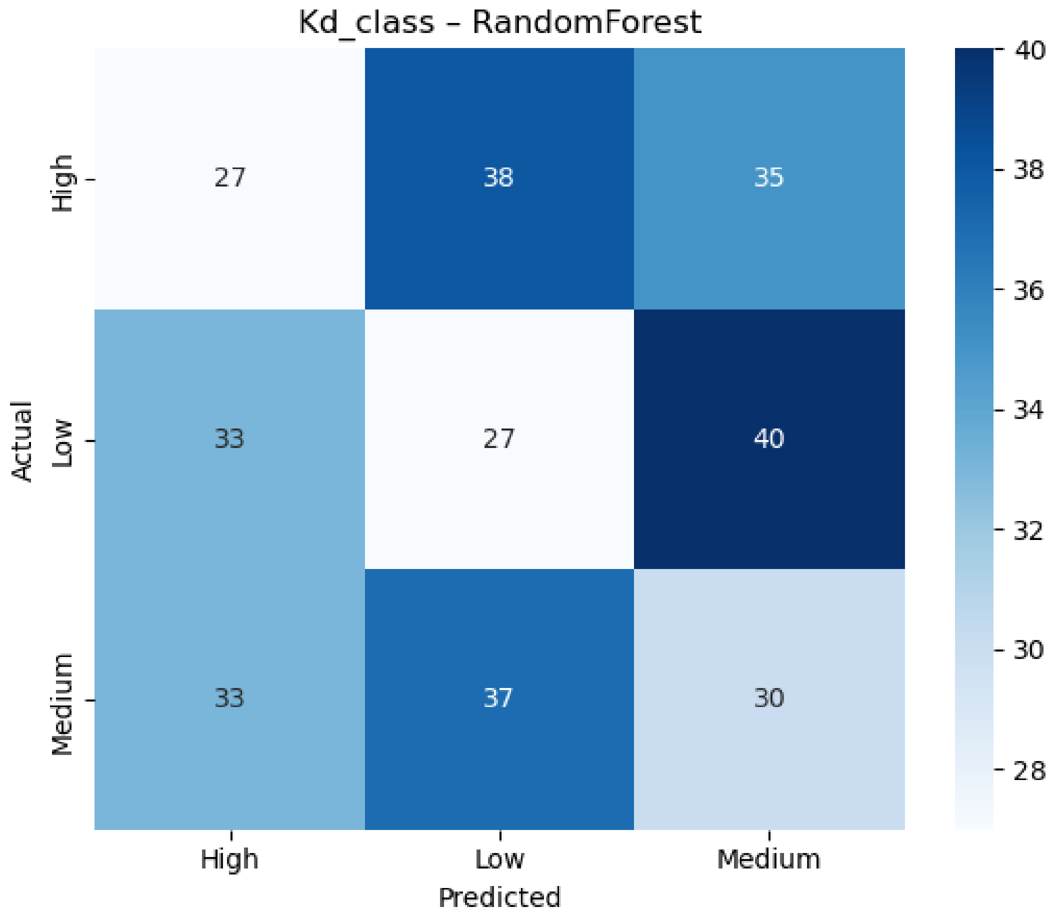

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Ki_class_LogisticRegression.png

Training DecisionTree → Ki_class
              precision    recall  f1-score   support

        High       0.32      0.28      0.30       100
         Low       0.32      0.33      0.33       100
      Medium       0.35      0.38      0.36       100

    accuracy                           0.33       300
   macro avg       0.33      0.33      0.33       300
weighted avg       0.33      0.33      0.33       300
```



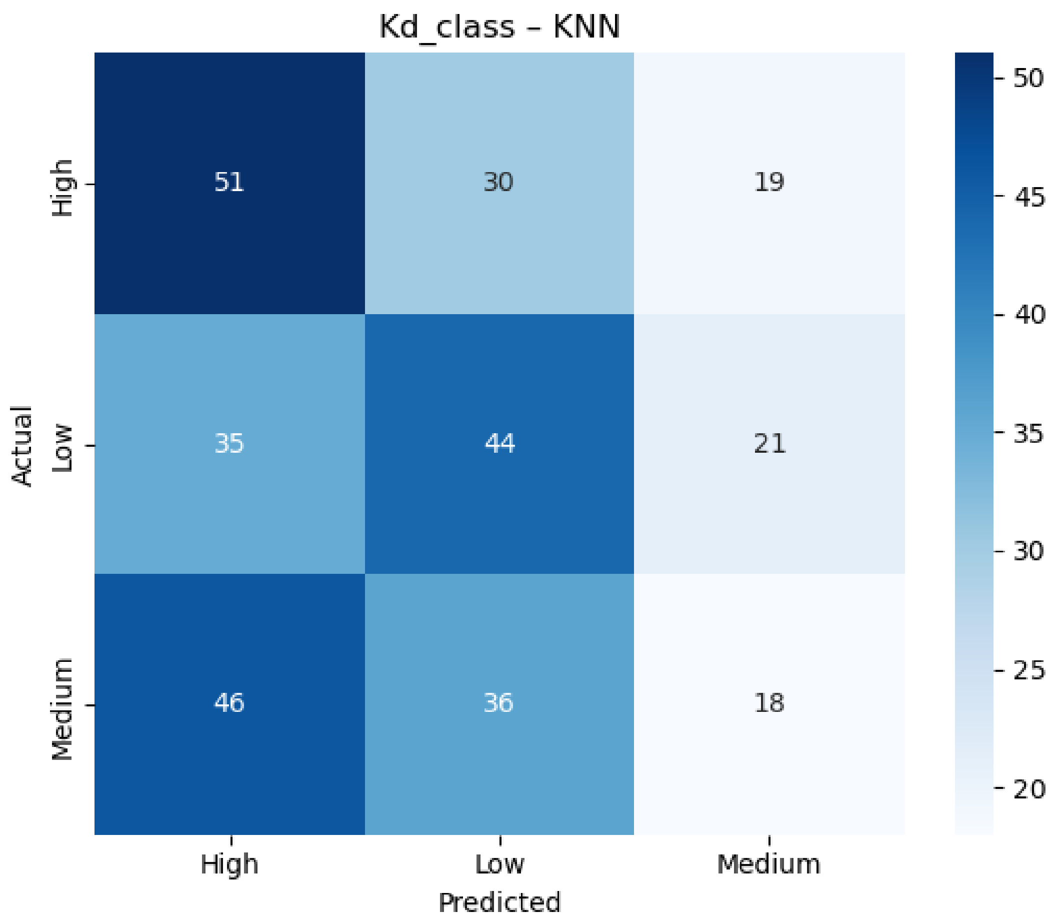

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Ki_class_DecisionTree.png

Training SVM → Ki_class
              precision    recall  f1-score   support

        High       0.30      0.37      0.33       100
         Low       0.28      0.27      0.28       100
      Medium       0.24      0.20      0.22       100

    accuracy                           0.28       300
   macro avg       0.28      0.28      0.28       300
weighted avg       0.28      0.28      0.28       300
```



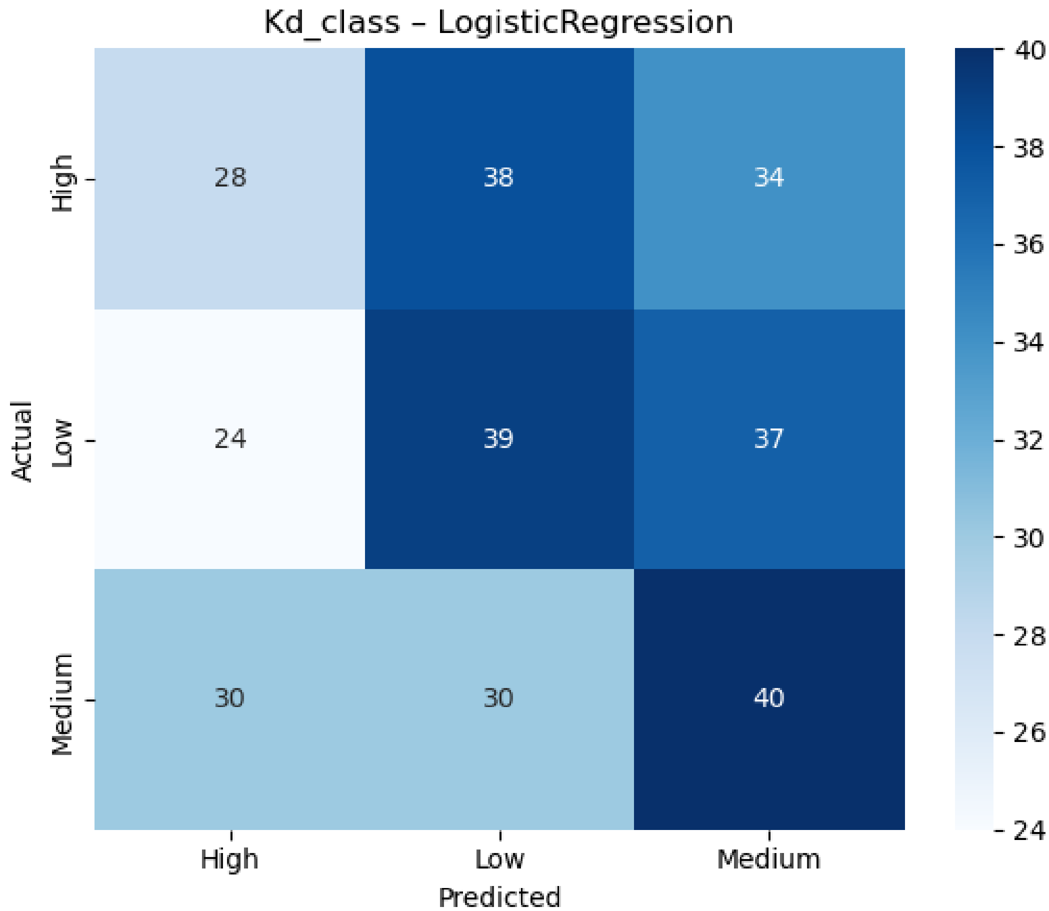

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Ki_class_SVM.png

======================================================================
PROCESSING TARGET: Kd_class
======================================================================

Training RandomForest → Kd_class
              precision    recall  f1-score   support

        High       0.29      0.27      0.28       100
         Low       0.26      0.27      0.27       100
      Medium       0.29      0.30      0.29       100

    accuracy                           0.28       300
   macro avg       0.28      0.28      0.28       300
weighted avg       0.28      0.28      0.28       300
```



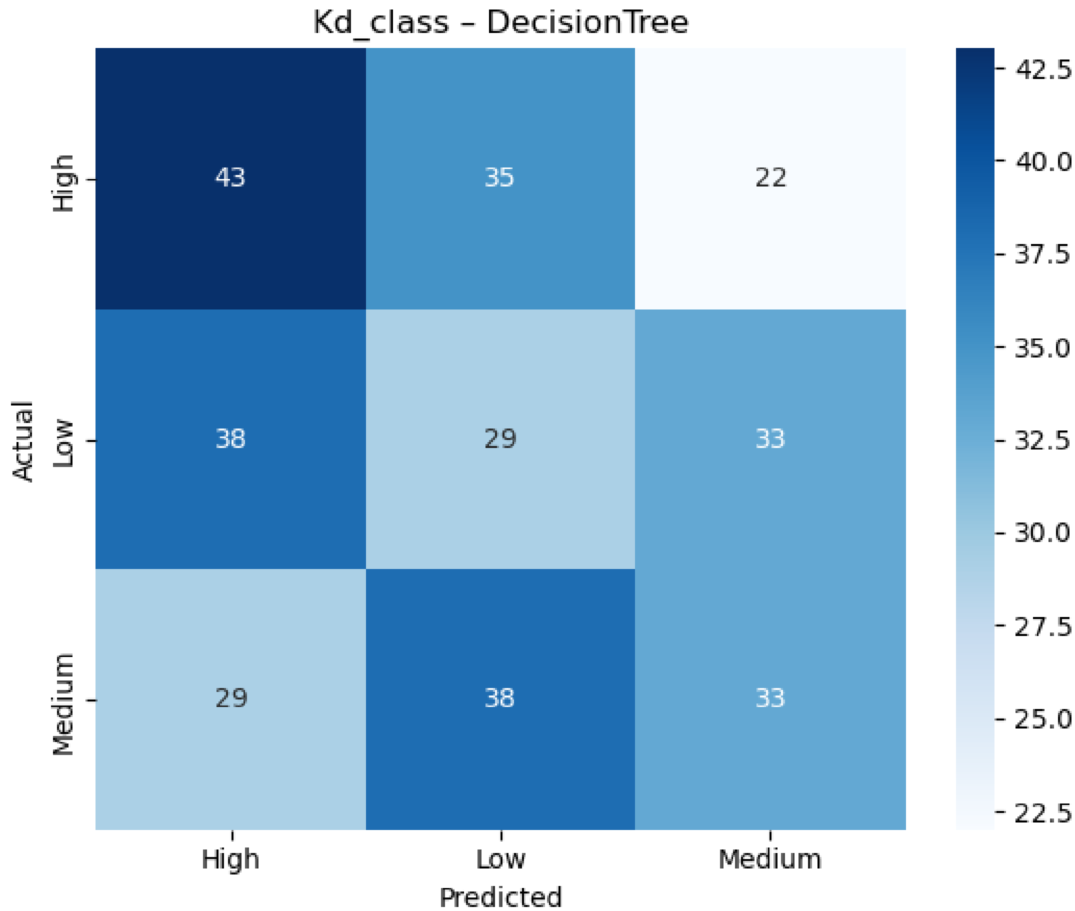

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kd_class_RandomForest.png

Training KNN → Kd_class
              precision    recall  f1-score   support

        High       0.39      0.51      0.44       100
         Low       0.40      0.44      0.42       100
      Medium       0.31      0.18      0.23       100

    accuracy                           0.38       300
   macro avg       0.37      0.38      0.36       300
weighted avg       0.37      0.38      0.36       300
```



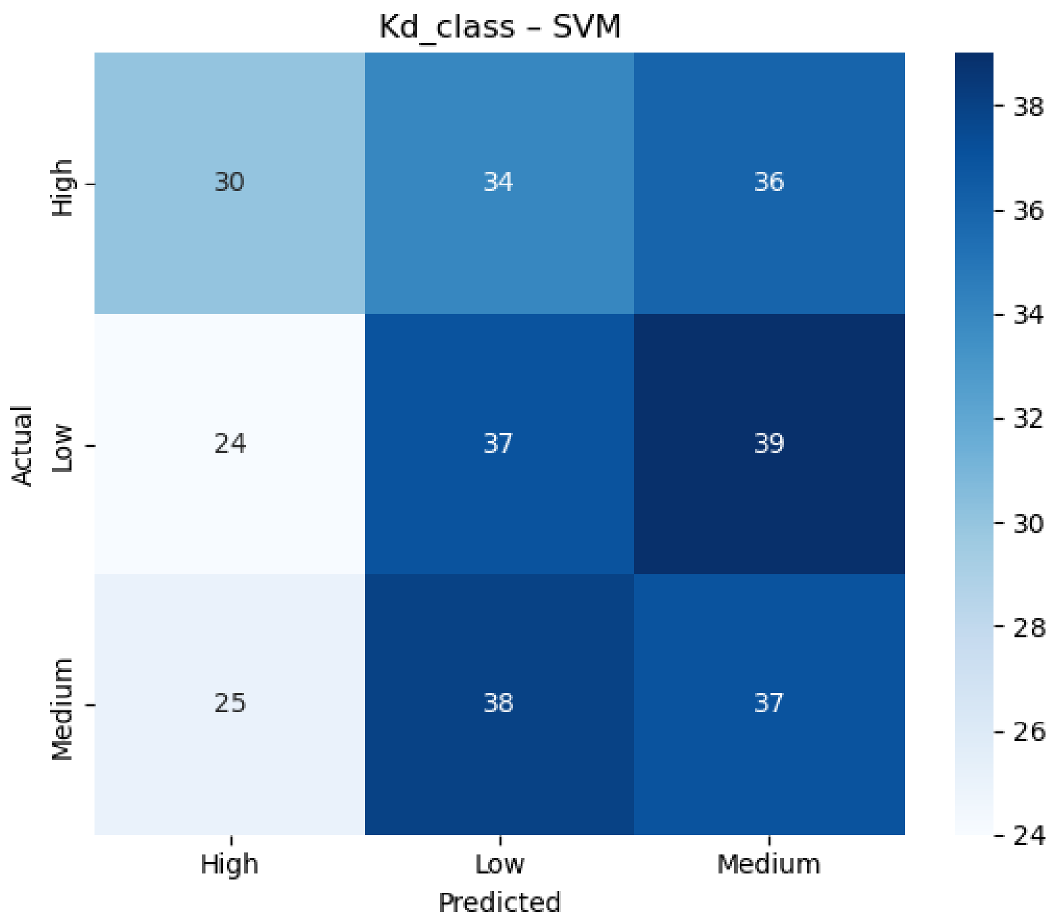

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kd_class_KNN.png

Training LogisticRegression → Kd_class
              precision    recall  f1-score   support

        High       0.34      0.28      0.31       100
         Low       0.36      0.39      0.38       100
      Medium       0.36      0.40      0.38       100

    accuracy                           0.36       300
   macro avg       0.36      0.36      0.35       300
weighted avg       0.36      0.36      0.35       300
```



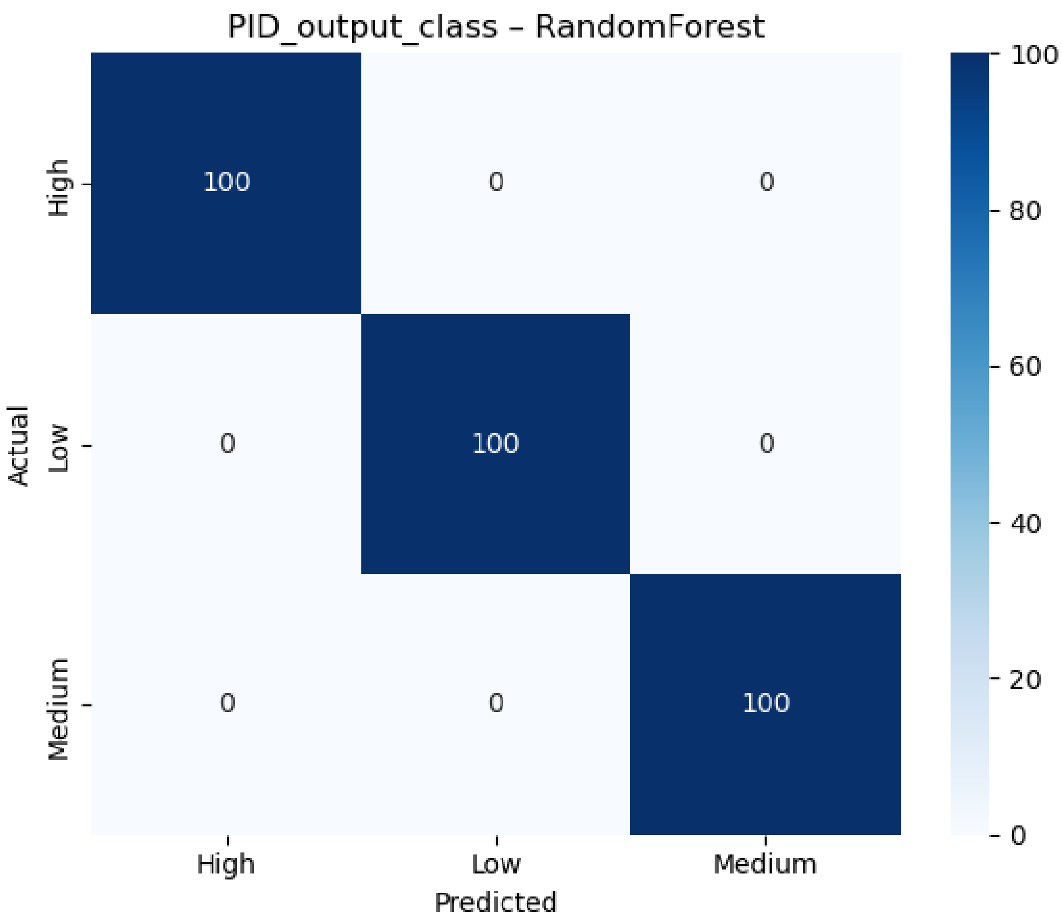

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Kd_class_LogisticRegression.png

Training DecisionTree → Kd_class
              precision    recall  f1-score   support

        High       0.39      0.43      0.41       100
         Low       0.28      0.29      0.29       100
      Medium       0.38      0.33      0.35       100

    accuracy                           0.35       300
   macro avg       0.35      0.35      0.35       300
weighted avg       0.35      0.35      0.35       300
```



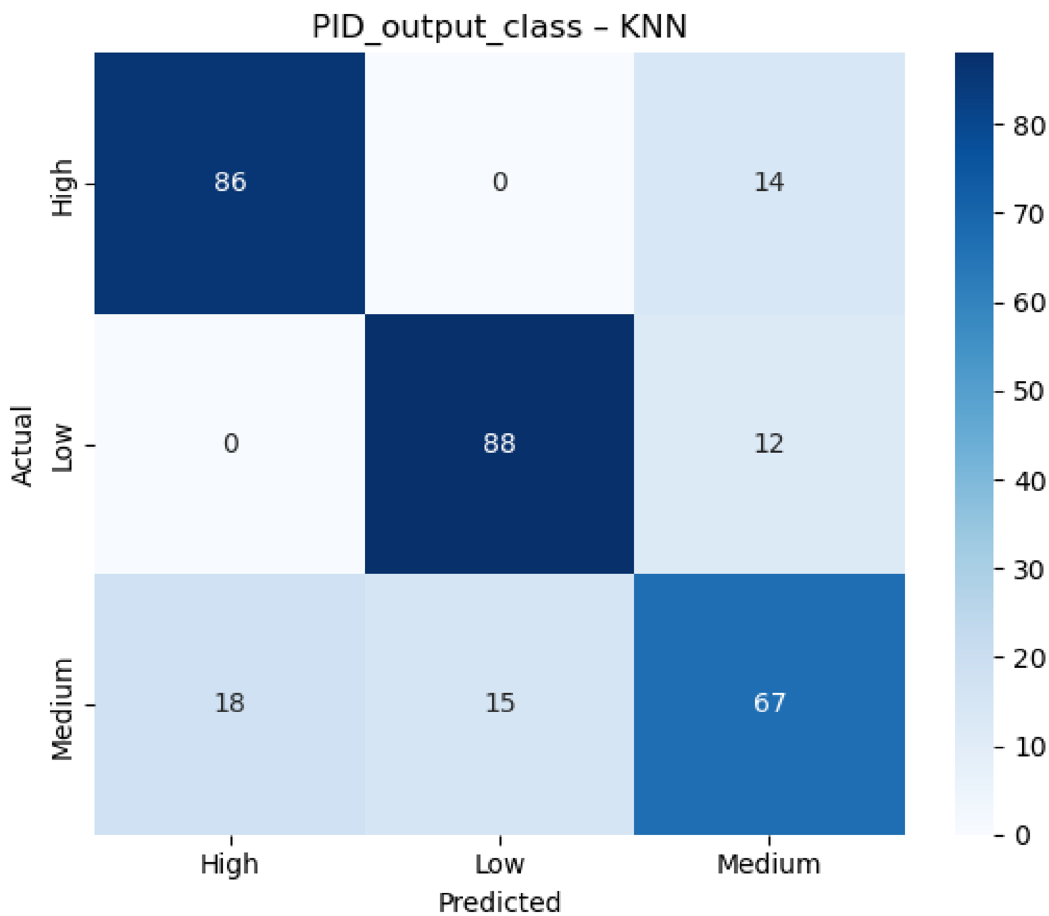

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kd_class_DecisionTree.png

Training SVM → Kd_class
              precision    recall  f1-score   support

        High       0.38      0.30      0.34       100
         Low       0.34      0.37      0.35       100
      Medium       0.33      0.37      0.35       100

    accuracy                           0.35       300
   macro avg       0.35      0.35      0.35       300
weighted avg       0.35      0.35      0.35       300
```



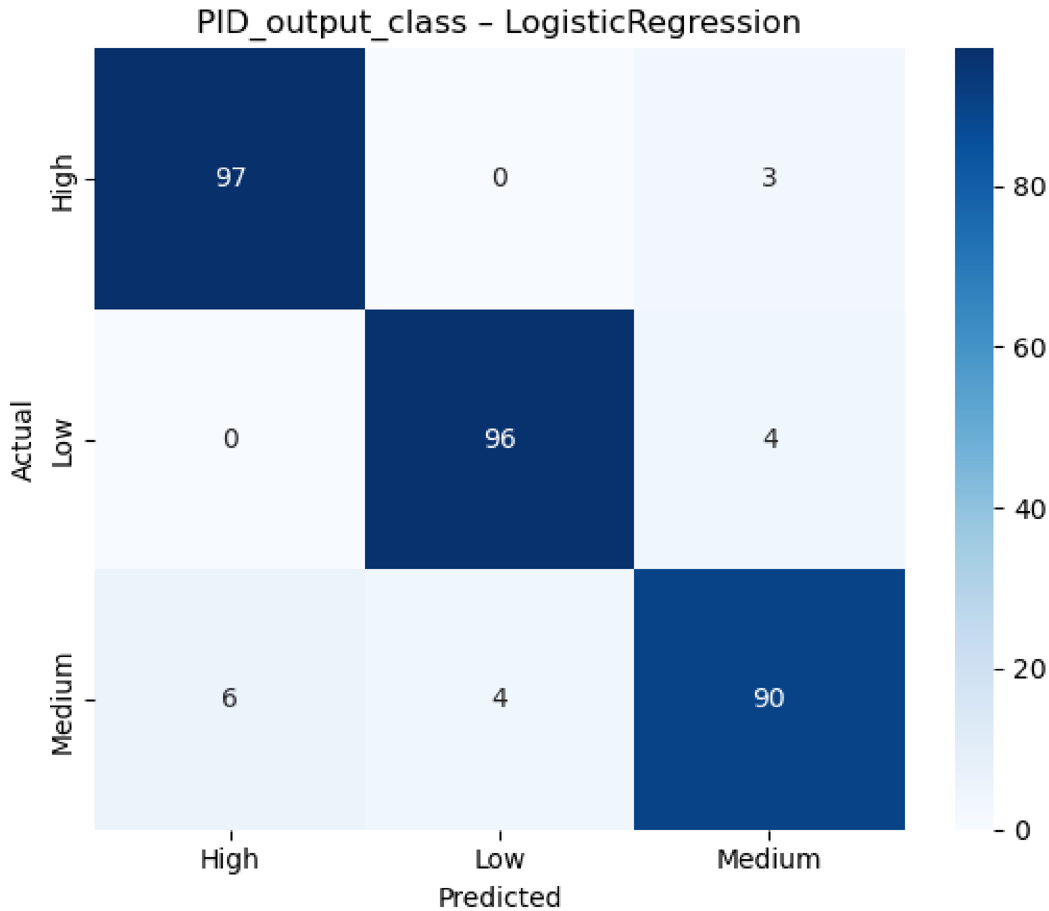

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_Kd_class_SVM.png

======================================================================
PROCESSING TARGET: PID_output_class
======================================================================

Training RandomForest → PID_output_class
              precision    recall  f1-score   support

        High       1.00      1.00      1.00       100
         Low       1.00      1.00      1.00       100
      Medium       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```



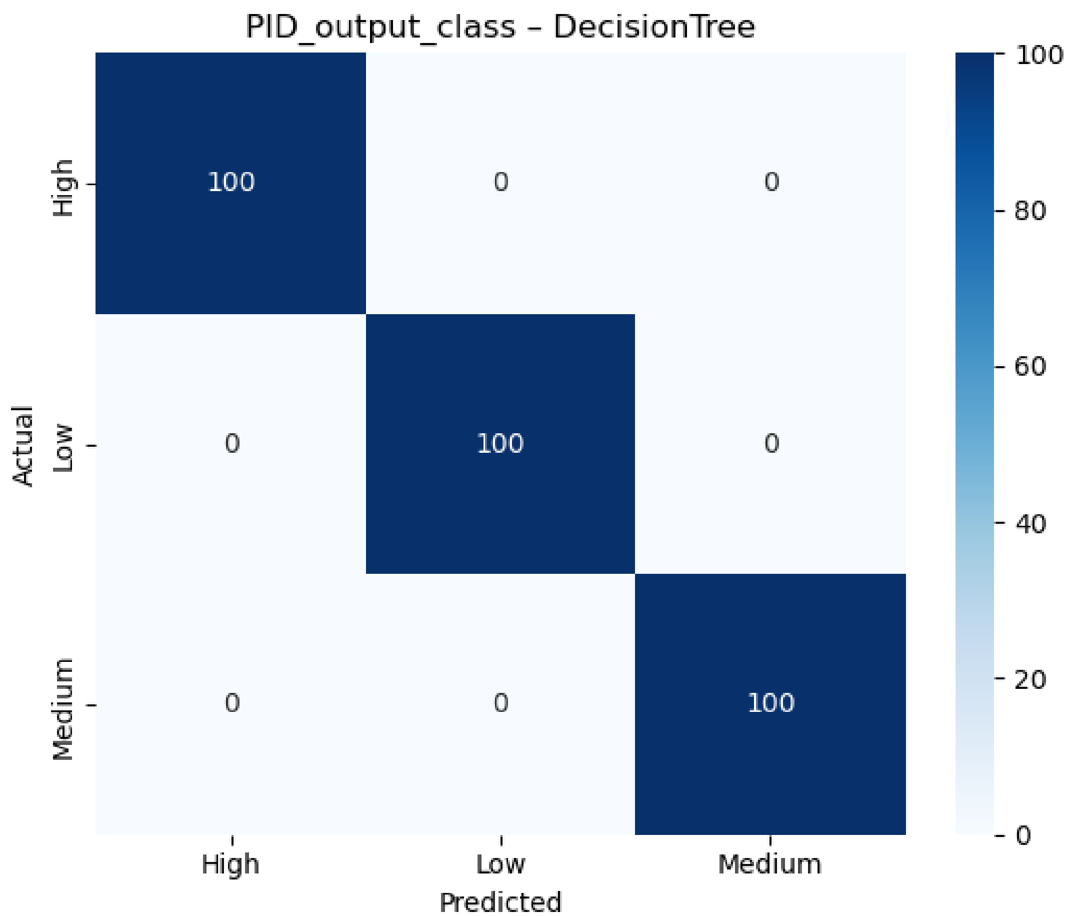

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_PID_output_class_RandomForest.png

Training KNN → PID_output_class
              precision    recall  f1-score   support

        High       0.83      0.86      0.84       100
         Low       0.85      0.88      0.87       100
      Medium       0.72      0.67      0.69       100

    accuracy                           0.80       300
   macro avg       0.80      0.80      0.80       300
weighted avg       0.80      0.80      0.80       300
```



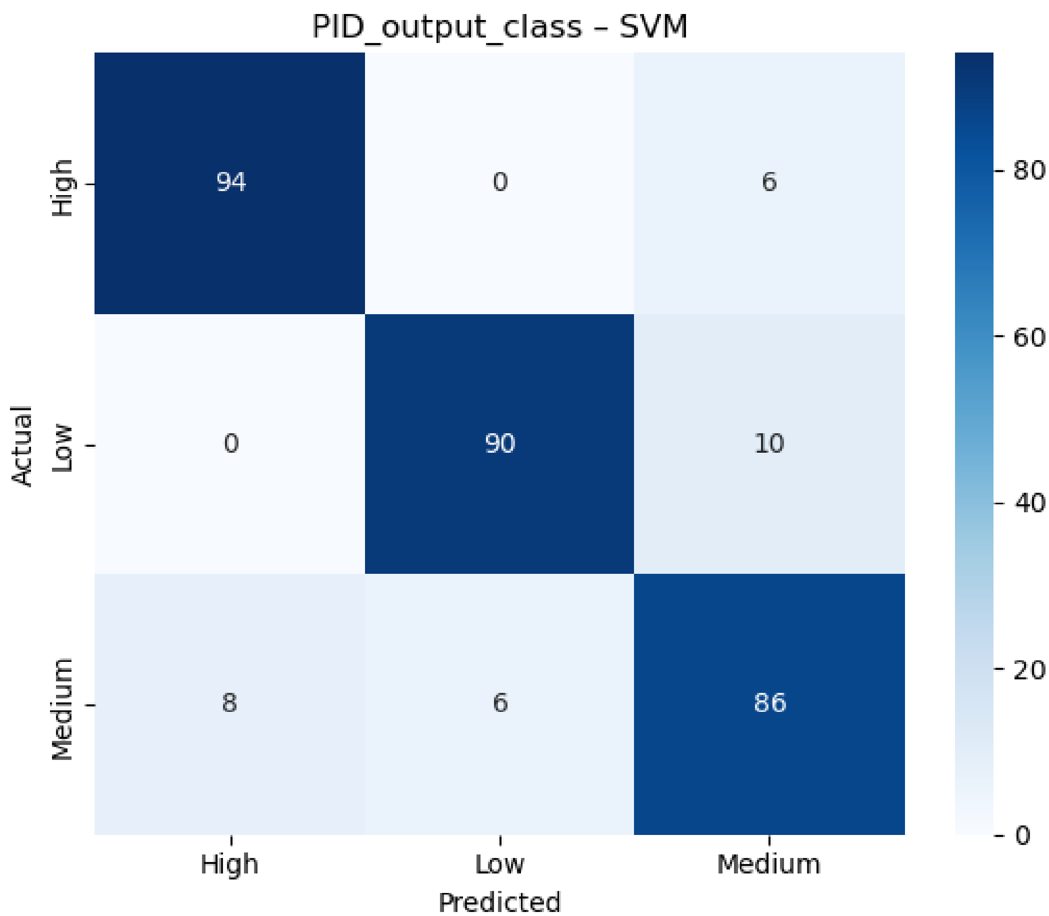

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_PID_output_class_KNN.png

Training LogisticRegression → PID_output_class
              precision    recall  f1-score   support

        High       0.94      0.97      0.96       100
         Low       0.96      0.96      0.96       100
      Medium       0.93      0.90      0.91       100

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```




```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_PID_output_class_LogisticRegression.png

Training DecisionTree → PID_output_class
              precision    recall  f1-score   support

        High       1.00      1.00      1.00       100
         Low       1.00      1.00      1.00       100
      Medium       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

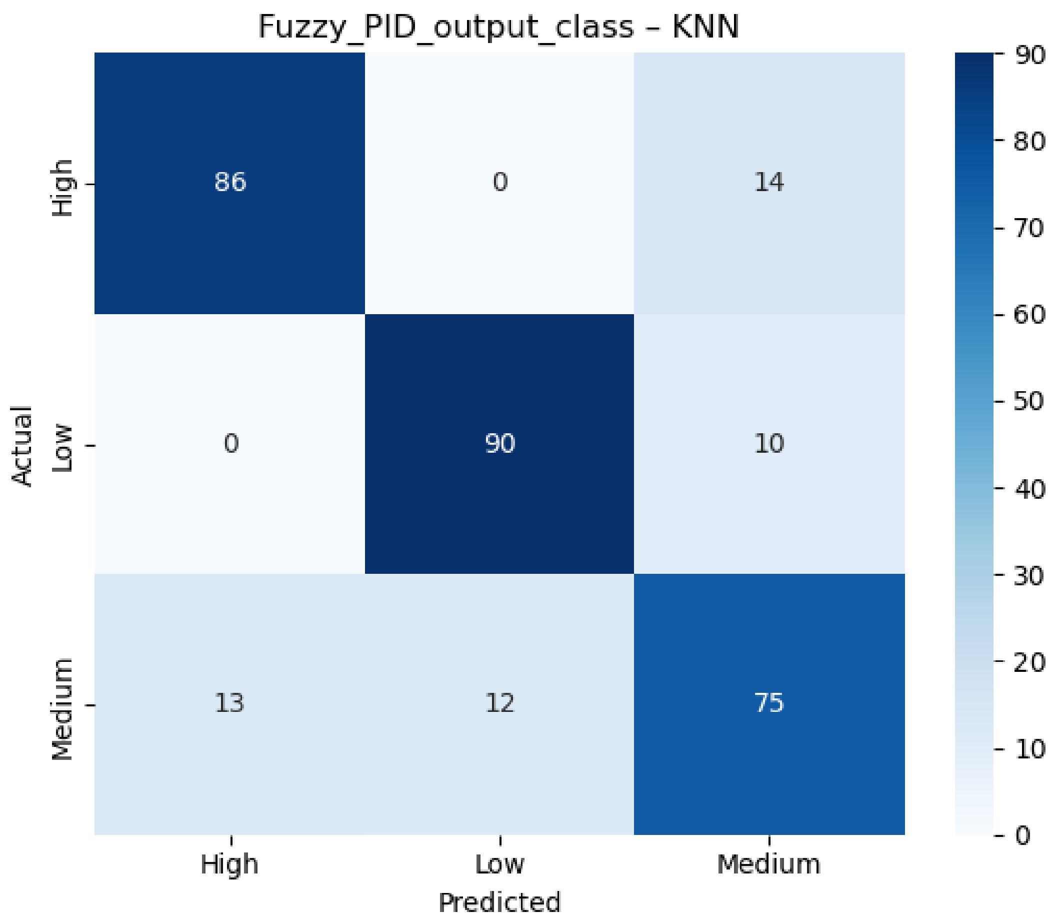



```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_PID_output_class_DecisionTree.png

Training SVM → PID_output_class
              precision    recall  f1-score   support

        High       0.92      0.94      0.93       100
         Low       0.94      0.90      0.92       100
      Medium       0.84      0.86      0.85       100

    accuracy                           0.90       300
   macro avg       0.90      0.90      0.90       300
weighted avg       0.90      0.90      0.90       300
```

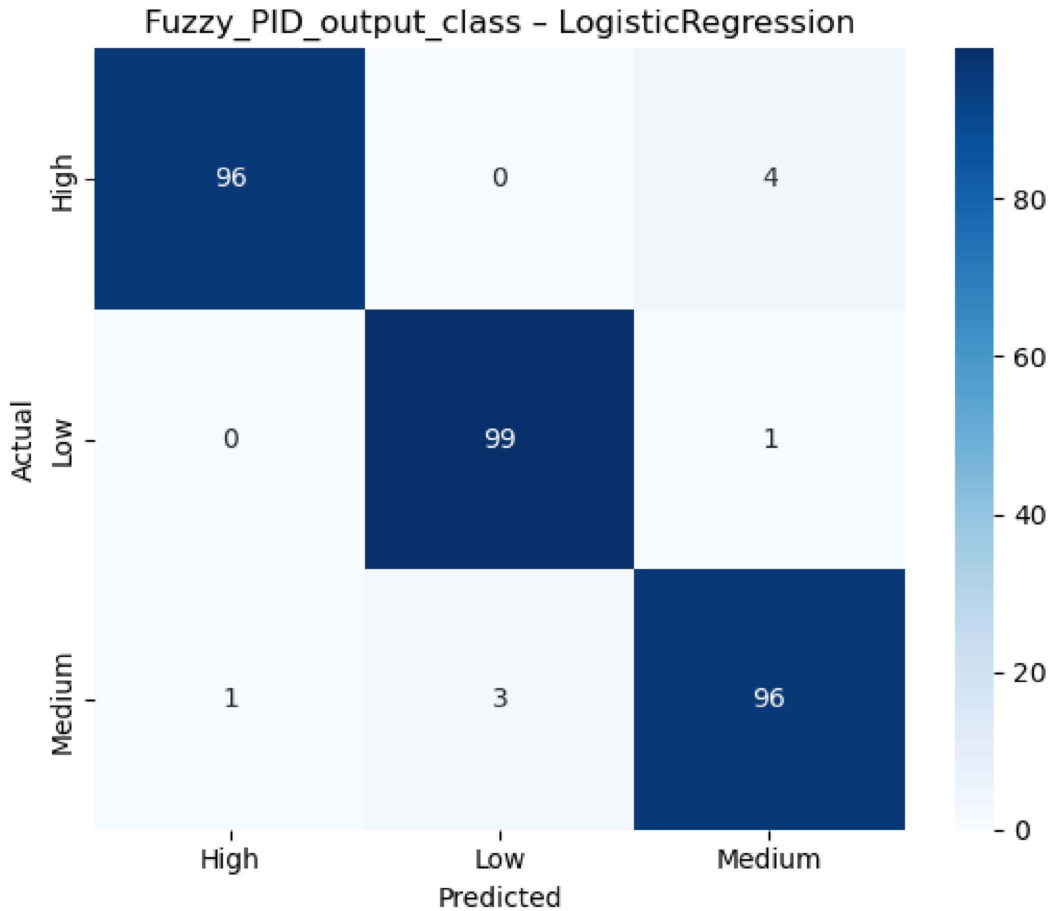



```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\CM_PID_output_class_SVM.png

======================================================================
PROCESSING TARGET: Fuzzy_PID_output_class
======================================================================

Training RandomForest → Fuzzy_PID_output_class
              precision    recall  f1-score   support

        High       1.00      0.98      0.99       100
         Low       1.00      0.99      0.99       100
      Medium       0.97      1.00      0.99       100

    accuracy                           0.99       300
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300
```

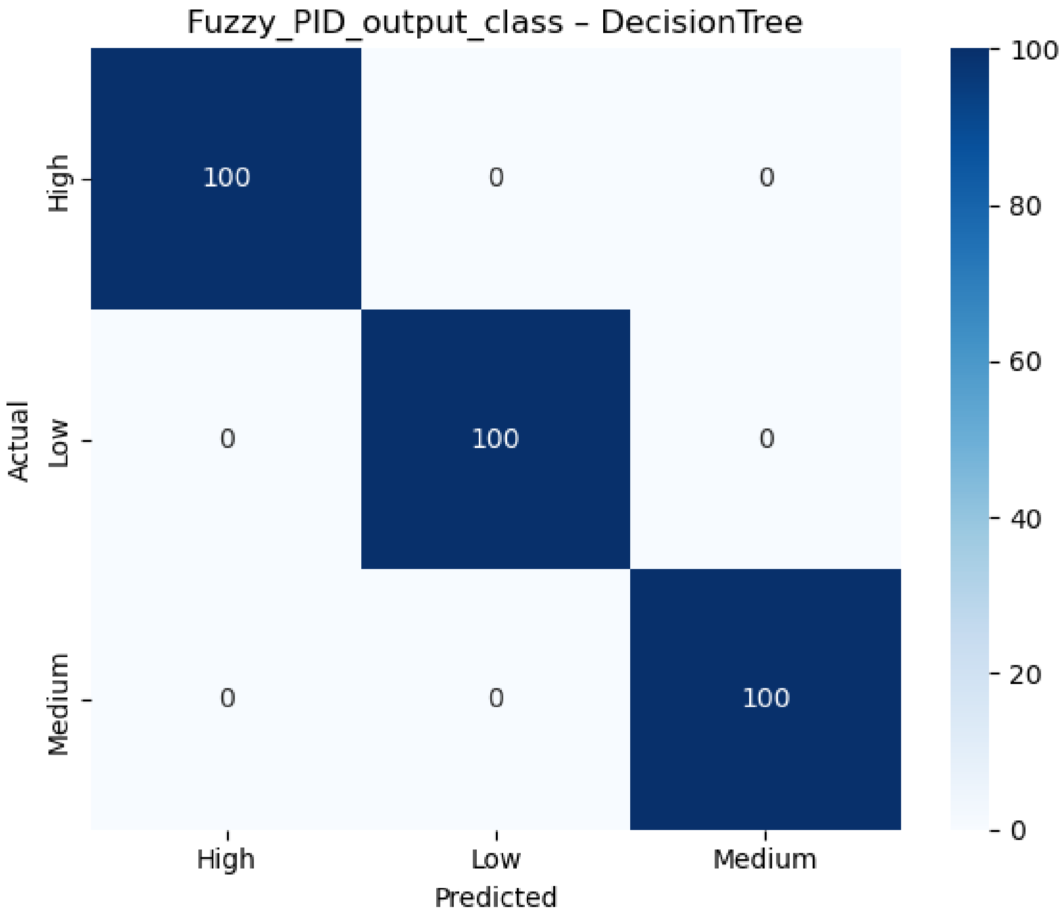

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Fuzzy_PID_output_class_RandomForest.png

Training KNN → Fuzzy_PID_output_class
              precision    recall  f1-score   support

        High       0.87      0.86      0.86       100
         Low       0.88      0.90      0.89       100
      Medium       0.76      0.75      0.75       100

    accuracy                           0.84       300
   macro avg       0.84      0.84      0.84       300
weighted avg       0.84      0.84      0.84       300
```



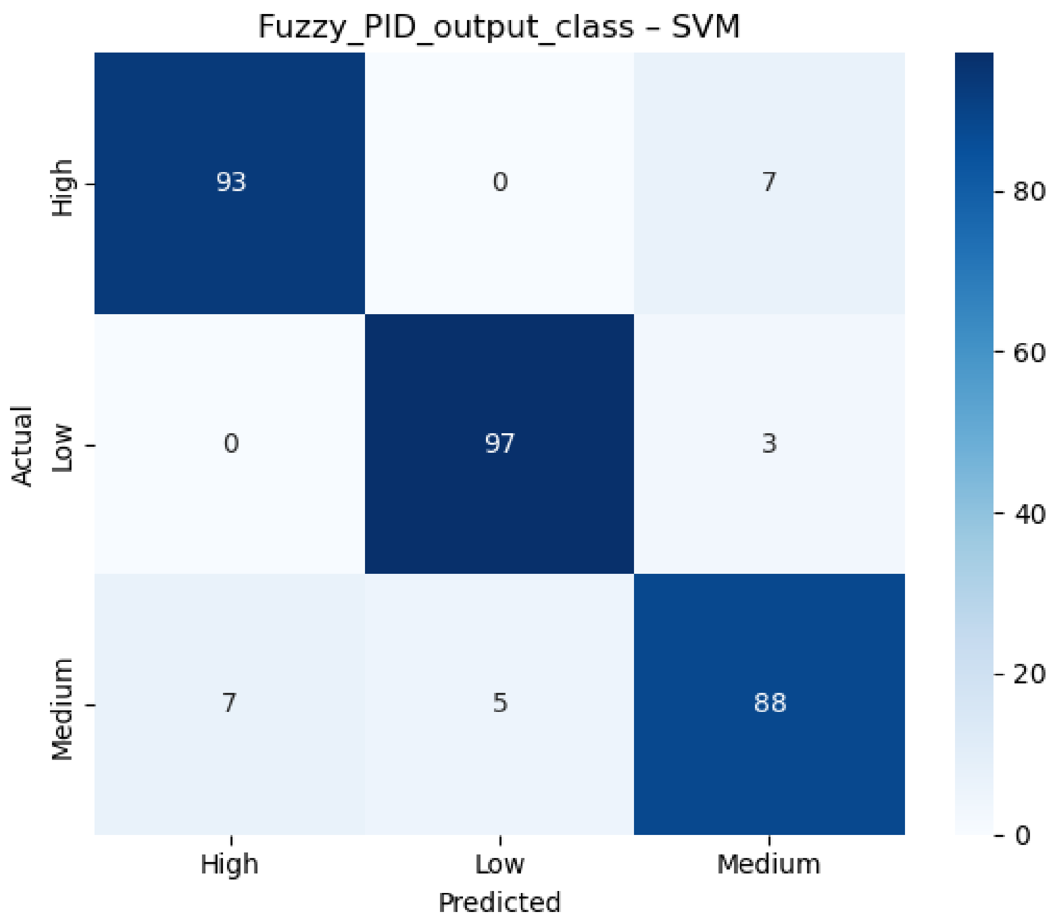

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Fuzzy_PID_output_class_KNN.png

Training LogisticRegression → Fuzzy_PID_output_class
              precision    recall  f1-score   support

        High       0.99      0.96      0.97       100
         Low       0.97      0.99      0.98       100
      Medium       0.95      0.96      0.96       100

    accuracy                           0.97       300
   macro avg       0.97      0.97      0.97       300
weighted avg       0.97      0.97      0.97       300
```




```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Fuzzy_PID_output_class_LogisticRegression.png

Training DecisionTree → Fuzzy_PID_output_class
              precision    recall  f1-score   support

        High       1.00      1.00      1.00       100
         Low       1.00      1.00      1.00       100
      Medium       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```




```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Fuzzy_PID_output_class_DecisionTree.png

Training SVM → Fuzzy_PID_output_class
              precision    recall  f1-score   support

        High       0.93      0.93      0.93       100
         Low       0.95      0.97      0.96       100
      Medium       0.90      0.88      0.89       100

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```



Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\CM_Fuzzy_PID_output_class_SVM.png

[19]:
```
# 8 Convert classification reports into metrics for plotting

def plot_model_metrics(model_names, accuracies, precisions, recalls, f1_scores,
                       title, out_dir, ylim=(0, 0.5)):

    x = np.arange(len(model_names))
    width = 0.2

    plt.figure(figsize=(12, 6))

    bars1 = plt.bar(x - width*1.5, accuracies, width, label="Accuracy")
    bars2 = plt.bar(x - width/2, precisions, width, label="Precision")
    bars3 = plt.bar(x + width/2, recalls, width, label="Recall")
    bars4 = plt.bar(x + width*1.5, f1_scores, width, label="F1-score")
```

```python
    # Add value labels
    for bars in [bars1, bars2, bars3, bars4]:
        for bar in bars:
            height = bar.get_height()
            plt.text(
                bar.get_x() + bar.get_width() / 2,
                height + 0.01,
                f"{height:.2f}",
                ha="center",
                va="bottom",
                fontsize=9
            )

    plt.xticks(x, model_names)
    plt.ylabel("Score")
    plt.title(title)
    plt.ylim(*ylim)
    plt.legend()
    plt.tight_layout()

    # Safe filename
    filename = title.replace(" ", "_").replace("-", "-") + ".png"
    save_path = os.path.join(out_dir, filename)
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.show()

    print("Saved:", save_path)
```

[21]:
```python
# 8.1 Bar chart for Kp, Ki, Kd, PID and Fuzzy-PID

all_models = {}
all_results = {}
all_metrics = {}

for target_name, y in classification_targets.items():

    print(f"\n--- Training for {target_name} ---")

    # Train-test split
    X_train, X_test, y_train, y_test_target = train_test_split(
        df[features], y, test_size=0.3, random_state=42, stratify=y
    )

    # Scaling
    scaler = StandardScaler()
    X_train_s = scaler.fit_transform(X_train)
```

```python
    X_test_s = scaler.transform(X_test)

    # Fresh model instances (IMPORTANT)
    models = {
        name: model.__class__(**model.get_params())
        for name, model in models_template.items()
    }

    target_results = {}

    # Train models
    for name, model in models.items():
        model.fit(X_train_s, y_train)
        preds = model.predict(X_test_s)
        target_results[name] = preds

    # Store
    all_models[target_name] = models
    all_results[target_name] = target_results

    # Compute metrics
    model_names, accuracies, precisions, recalls, f1_scores = [], [], [], [], []

    for name, preds in target_results.items():
        acc = accuracy_score(y_test_target, preds)
        report = classification_report(y_test_target, preds, output_dict=True)

        model_names.append(name)
        accuracies.append(acc)
        precisions.append(report["macro avg"]["precision"])
        recalls.append(report["macro avg"]["recall"])
        f1_scores.append(report["macro avg"]["f1-score"])

    all_metrics[target_name] = {
        "model_names": model_names,
        "accuracies": accuracies,
        "precisions": precisions,
        "recalls": recalls,
        "f1_scores": f1_scores
    }

    # Plot and save
    plot_model_metrics(
        model_names,
        accuracies,
        precisions,
        recalls,
```

```
        f1_scores,
        title=f"Model Comparison - {target_name}",
        out_dir=OUT_DIR,
        ylim=(0, 0.6)
    )
```

```
--- Training for Kp_class ---
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[21], line 24
     19 X_test_s = scaler.transform(X_test)
     21 # Fresh model instances (IMPORTANT)
     22 models = {
     23     name: model.__class__(**model.get_params())
---> 24     for name, model in models_template.items()
     25 }
     27 target_results = {}
     29 # Train models

NameError: name 'models_template' is not defined
```

[ ]:

```
# 8.2 Plot heatmap
# Build a metrics table for all models

# Loop through all targets and create heatmaps
for target_name, metrics in all_metrics.items():
    print(f"\n--- Generating heatmap for {target_name} ---")

    # Build metrics table
    metrics_table = pd.DataFrame({
        "Accuracy": metrics["accuracies"],
        "Precision (macro)": metrics["precisions"],
        "Recall (macro)": metrics["recalls"],
        "F1-score (macro)": metrics["f1_scores"]
    }, index=metrics["model_names"])

    print(f"\nMetrics Table for {target_name}:")
    print(metrics_table)

    # Plot heatmap
    plt.figure(figsize=(10, 6))
    sns.heatmap(metrics_table, annot=True, cmap="viridis", fmt=".3f")
    plt.title(f"Model Performance Heatmap - {target_name}")
    plt.ylabel("Model")
```

```
    plt.xlabel("Metric")
    plt.tight_layout()

    # Save figure
    save_path = os.path.join(OUT_DIR, f"Model Performance Heatmap -␣
↪{target_name}.png")
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.show()
```

[ ]:
```
# 8.3 COMBINE ALL 5 BAR CHARTS INTO ONE MULTI-PANEL FIGURE

def plot_combined_metrics(all_metrics, out_dir, ylim=(0, 0.6)):

    targets = list(all_metrics.keys())
    n_targets = len(targets)

    fig, axes = plt.subplots(n_targets, 1, figsize=(14, 4 * n_targets),␣
↪sharex=True)

    if n_targets == 1:
        axes = [axes]

    for ax, target in zip(axes, targets):

        m = all_metrics[target]
        model_names = m["model_names"]
        accuracies = m["accuracies"]
        precisions = m["precisions"]
        recalls = m["recalls"]
        f1_scores = m["f1_scores"]

        x = np.arange(len(model_names))
        width = 0.2

        ax.bar(x - width*1.5, accuracies, width, label="Accuracy")
        ax.bar(x - width/2, precisions, width, label="Precision")
        ax.bar(x + width/2, recalls, width, label="Recall")
        ax.bar(x + width*1.5, f1_scores, width, label="F1-score")

        ax.set_title(f"{target}")
        ax.set_ylabel("Score")
        ax.set_ylim(*ylim)
        ax.grid(axis="y", alpha=0.3)

        # Value labels
        for values, offset in zip(
            [accuracies, precisions, recalls, f1_scores],
```

```
            [-1.5, -0.5, 0.5, 1.5]
        ):
            for i, v in enumerate(values):
                ax.text(i + width*offset, v + 0.01, f"{v:.2f}",
                        ha="center", va="bottom", fontsize=8)

    axes[-1].set_xticks(x)
    axes[-1].set_xticklabels(model_names)

    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc="upper center", ncol=4)

    plt.tight_layout(rect=[0, 0, 1, 0.97])

    save_path = os.path.join(out_dir, "Combined_Model_Comparison_All_Targets.
↪png")
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.show()

    print("Saved combined plot →", save_path)
```

[25]:
```
plot_combined_metrics(all_metrics, OUT_DIR)
save_path = os.path.join(OUT_DIR, f"Combined Accuracy Precision Recall F1 score␣
↪plots.png")
plt.savefig(save_path, dpi=300, bbox_inches='tight')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[25], line 1
----> 1 plot_combined_metrics(all_metrics, OUT_DIR)
      2 save_path = os.path.join(OUT_DIR, f"Combined Accuracy Precision Recall␣
↪F1 score plots.png")
      3 plt.savefig(save_path, dpi=300, bbox_inches='tight')

NameError: name 'plot_combined_metrics' is not defined
```

[27]:
```
# 8.4 STATISTICAL COMPARISON TABLE

expected_targets = [
    "Kp_class",
    "Ki_class",
    "Kd_class",
    "PID_output_class",
    "Fuzzy_PID_output_class"
]
```

```
# Sanity check
missing = set(expected_targets) - set(all_metrics.keys())
if missing:
    raise ValueError(f"Missing targets in all_metrics: {missing}")

comparison_rows = []

for target in expected_targets:
    metrics = all_metrics[target]

    for i, model in enumerate(metrics["model_names"]):
        comparison_rows.append({
            "Target": target,
            "Model": model,
            "Accuracy": metrics["accuracies"][i],
            "Precision": metrics["precisions"][i],
            "Recall": metrics["recalls"][i],
            "F1_score": metrics["f1_scores"][i]
        })

comparison_df = pd.DataFrame(comparison_rows)

# Save table
table_path = os.path.join(OUT_DIR, "Model_Comparison_Table_All_5_Targets.csv")
comparison_df.to_csv(table_path, index=False)

print("Saved comparison table →", table_path)
display(comparison_df.head(10))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[27], line 14
     12 missing = set(expected_targets) - set(all_metrics.keys())
     13 if missing:
---> 14     raise ValueError(f"Missing targets in all_metrics: {missing}")
     16 comparison_rows = []
     18 for target in expected_targets:

ValueError: Missing targets in all_metrics: {'PID_output_class', 'Kd_class',␣
↪'Fuzzy_PID_output_class', 'Ki_class', 'Kp_class'}
```

[29]:
```
# 8.5 RANK MODELS PER TARGET (BASED ON F1-SCORE)

ranking_tables = {}
```

```python
for target, metrics in all_metrics.items():

    rank_df = pd.DataFrame({
        "Model": metrics["model_names"],
        "Accuracy": metrics["accuracies"],
        "Precision": metrics["precisions"],
        "Recall": metrics["recalls"],
        "F1_score": metrics["f1_scores"]
    })

    rank_df = rank_df.sort_values(by="F1_score", ascending=False)
    rank_df["Rank"] = range(1, len(rank_df) + 1)

    ranking_tables[target] = rank_df

    # Save per-target ranking
    rank_path = os.path.join(OUT_DIR, f"Model_Ranking_{target}.csv")
    rank_df.to_csv(rank_path, index=False)

    print(f"\nModel Ranking for {target}")
    display(rank_df)
```

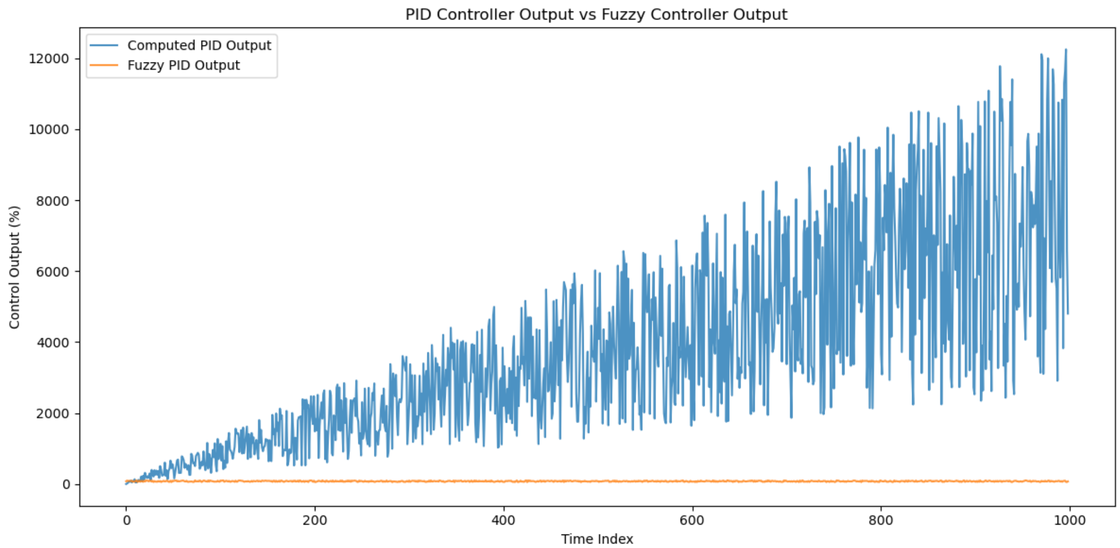
[31]:

```python
# 9 Compute PID controller output u(t)
# PID equation:
# u(t) = Kp * e(t) + Ki *  e(t) dt + Kd * de(t)/dt

df["Integral_Error"] = df["Temperature Error (°C)"].cumsum()
df["Derivative_Error"] = df["Temperature Error (°C)"].diff().fillna(0)

df["PID_Output_Computed"] = (
    df["PID Kp"] * df["Temperature Error (°C)"] +
    df["PID Ki"] * df["Integral_Error"] +
    df["PID Kd"] * df["Derivative_Error"]
)
```

[33]:

```python
# 10 Compare PID vs Fuzzy Controller
plt.figure(figsize=(12, 6))
plt.plot(df["PID_Output_Computed"], label="Computed PID Output", alpha=0.8)
plt.plot(df["Fuzzy PID Control Output (%)"], label="Fuzzy PID Output", alpha=0.
 ↪8)
plt.title("PID Controller Output vs Fuzzy Controller Output")
plt.xlabel("Time Index")
plt.ylabel("Control Output (%)")
plt.legend()
plt.tight_layout()
plot_path = os.path.join(OUT_DIR, "pid_vs_fuzzy_output.png")
plt.savefig(plot_path, dpi=300, bbox_inches="tight")
```

```
plt.show()
```



0.0.1 Predict Actual Gains

[166]:
```
# Regression Pipeline for Predicting Kp, Ki, Kd
```

[36]:
```
# XGBRegressor
try:
    from xgboost import XGBRegressor
    xgb_available = True
except ImportError:
    xgb_available = False
```

XGBoost (Extreme Gradient Boosting) is a highly efficient and widely used machine learning algorithm designed for supervised learning tasks such as regression, classification, and ranking. It builds on the concept of gradient boosting, where multiple weak learners—typically decision trees—are combined sequentially to create a strong predictive model. Each new tree focuses on correcting the errors of the previous trees, which allows the model to achieve high accuracy. Additionally, XGBoost incorporates L1 and L2 regularization to penalize overly complex models, reducing the risk of overfitting and improving the generalization of predictions on unseen data. Its ability to handle missing values natively further enhances its flexibility and ease of use in practical applications.

Beyond accuracy, XGBoost is highly valued for its computational efficiency and scalability. The algorithm is optimized for speed, supports parallel processing, and can leverage both CPU and GPU resources, making it suitable for large datasets. It also offers the ability to customize objective functions and evaluation metrics according to specific problem requirements. Another key advantage is its ability to provide feature importance insights, helping practitioners understand which variables contribute most to predictions. These qualities make XGBoost a preferred choice in many real-world applications, particularly when working with structured or tabular data where performance and interpretability are both important.

[39]:
```
regression_targets = {
    "Kp": df["PID Kp"],
    "Ki": df["PID Ki"],
    "Kd": df["PID Kd"]
}
```

[41]:
```
reg_models = {
    "RandomForest": RandomForestRegressor(
        n_estimators=300, random_state=42, n_jobs=-1
    ),
    "NeuralNetwork": MLPRegressor(
        hidden_layer_sizes=(64, 32),
        activation="relu",
        max_iter=1000,
        random_state=42
    )
}

if xgb_available:
    reg_models["XGBoost"] = XGBRegressor(
        n_estimators=300,
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )
```

[43]:
```
regression_results = []
regression_predictions = {}

for target_name, y in regression_targets.items():

    print("\n" + "="*60)
    print(f"REGRESSION TARGET: {target_name}")
    print("="*60)

    X_train, X_test, y_train, y_test = train_test_split(
        df[features], y, test_size=0.3, random_state=42
    )

    scaler = StandardScaler()
    X_train_s = scaler.fit_transform(X_train)
    X_test_s = scaler.transform(X_test)
```

```python
    regression_predictions[target_name] = {}

    for model_name, model in reg_models.items():

        print(f"Training {model_name}...")

        model.fit(X_train_s, y_train)
        preds = model.predict(X_test_s)

        rmse = np.sqrt(mean_squared_error(y_test, preds))
        mae = mean_absolute_error(y_test, preds)
        r2 = r2_score(y_test, preds)

        regression_results.append({
            "Target": target_name,
            "Model": model_name,
            "RMSE": rmse,
            "MAE": mae,
            "R2": r2
        })

        regression_predictions[target_name][model_name] = (y_test, preds)

        print(f"RMSE={rmse:.4f}, MAE={mae:.4f}, R2={r2:.4f}")
```

```
============================================================
REGRESSION TARGET: Kp
============================================================
Training RandomForest…
RMSE=0.2830, MAE=0.2411, R2=-0.0634
Training NeuralNetwork…
RMSE=0.3417, MAE=0.2803, R2=-0.5502
Training XGBoost…
RMSE=0.2947, MAE=0.2480, R2=-0.1529

============================================================
REGRESSION TARGET: Ki
============================================================
Training RandomForest…
RMSE=0.1234, MAE=0.1066, R2=-0.1226
Training NeuralNetwork…
RMSE=0.1371, MAE=0.1155, R2=-0.3848
Training XGBoost…
RMSE=0.1274, MAE=0.1091, R2=-0.1972

============================================================
REGRESSION TARGET: Kd
```

```
============================================================
Training RandomForest…
RMSE=0.1218, MAE=0.1041, R2=-0.0526
Training NeuralNetwork…
RMSE=0.1424, MAE=0.1190, R2=-0.4402
Training XGBoost…
RMSE=0.1262, MAE=0.1074, R2=-0.1309
```

[45]:
```
regression_df = pd.DataFrame(regression_results)

reg_table_path = os.path.join(OUT_DIR, "PID_Gain_Regression_Metrics.csv")
regression_df.to_csv(reg_table_path, index=False)

print("Saved regression metrics →", reg_table_path)
display(regression_df)
```

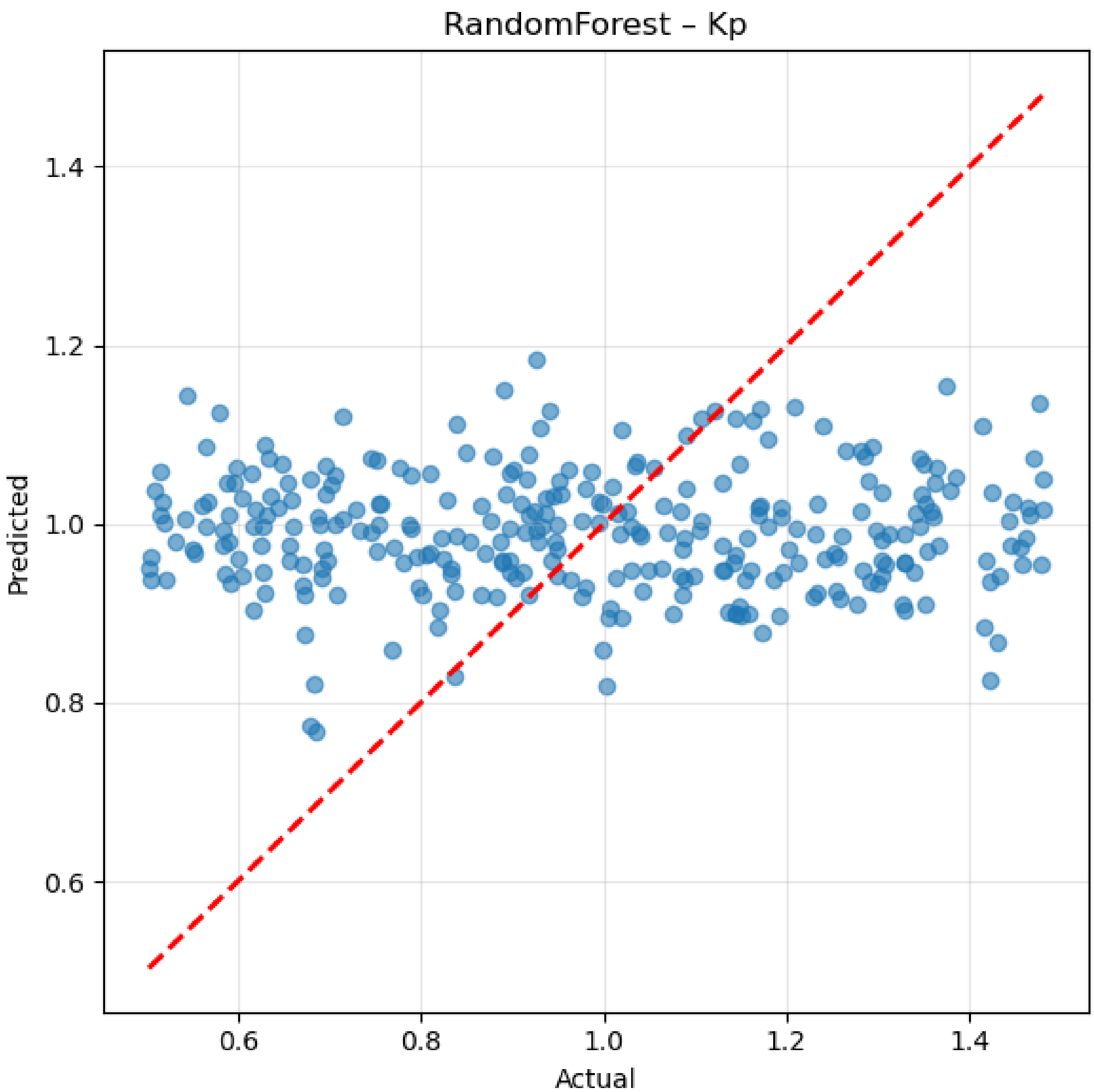
```
Saved regression metrics → I:\Self Study\python study\A Practical Industrial ML
Applications for Smart Manufacturing Temperature
Regulation\outputs\PID_Gain_Regression_Metrics.csv
```

	Target	Model	RMSE	MAE	R2
0	Kp	RandomForest	0.282997	0.241050	-0.063437
1	Kp	NeuralNetwork	0.341680	0.280257	-0.550193
2	Kp	XGBoost	0.294661	0.248000	-0.152906
3	Ki	RandomForest	0.123399	0.106606	-0.122640
4	Ki	NeuralNetwork	0.137051	0.115517	-0.384777
5	Ki	XGBoost	0.127433	0.109119	-0.197237
6	Kd	RandomForest	0.121761	0.104089	-0.052561
7	Kd	NeuralNetwork	0.142427	0.118991	-0.440172
8	Kd	XGBoost	0.126210	0.107421	-0.130882

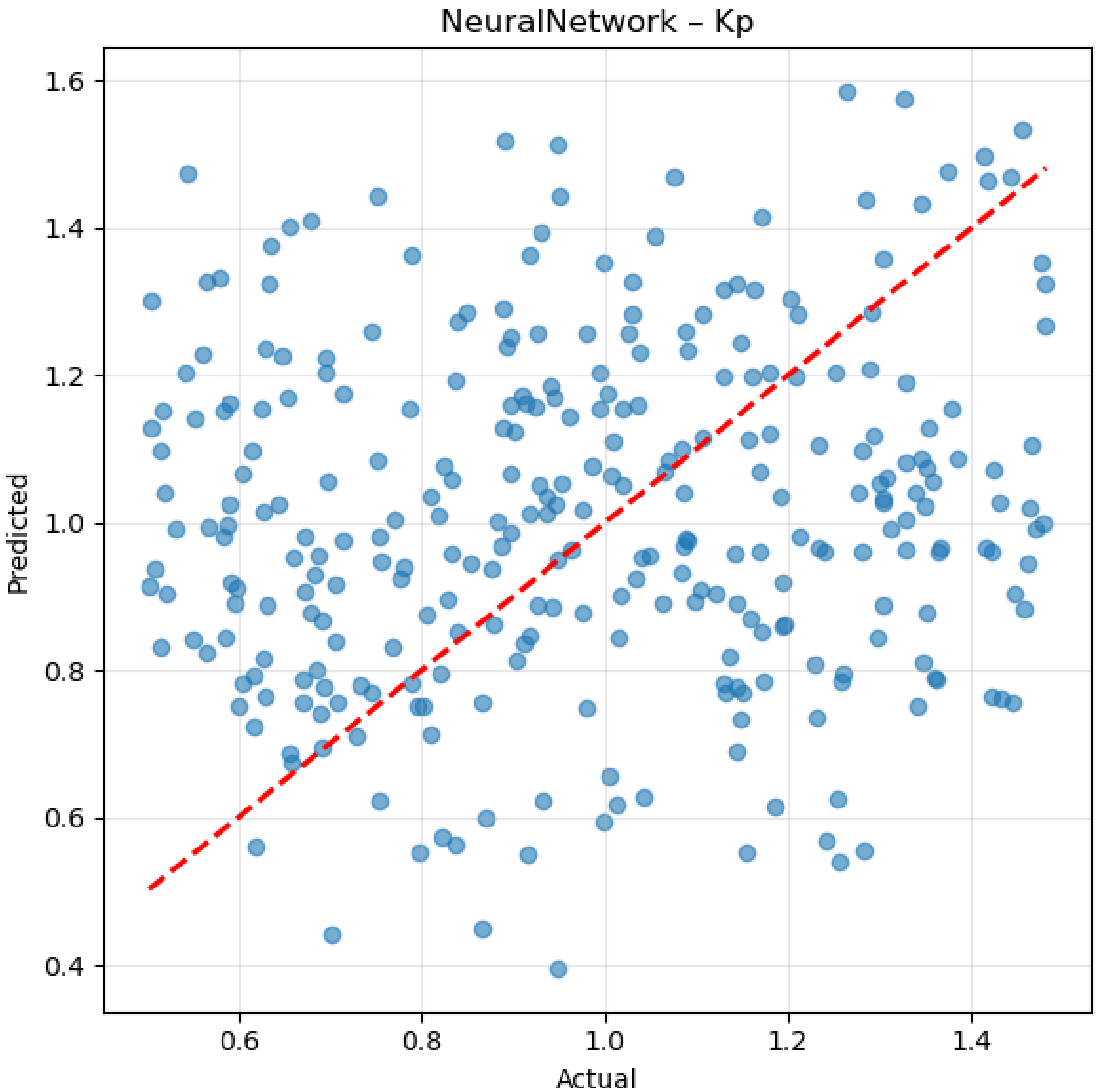
[47]:
```
def plot_predicted_vs_actual(y_true, y_pred, target, model, out_dir):

    plt.figure(figsize=(6, 6))
    plt.scatter(y_true, y_pred, alpha=0.6)
    plt.plot(
        [y_true.min(), y_true.max()],
        [y_true.min(), y_true.max()],
        "r--", linewidth=2
    )

    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"{model} - {target}")
    plt.grid(alpha=0.3)
    plt.tight_layout()

    save_path = os.path.join(out_dir, f"Pred_vs_Actual_{target}_{model}.png")
```

```
    plt.savefig(save_path, dpi=300)
    plt.show()

    print("Saved:", save_path)
```

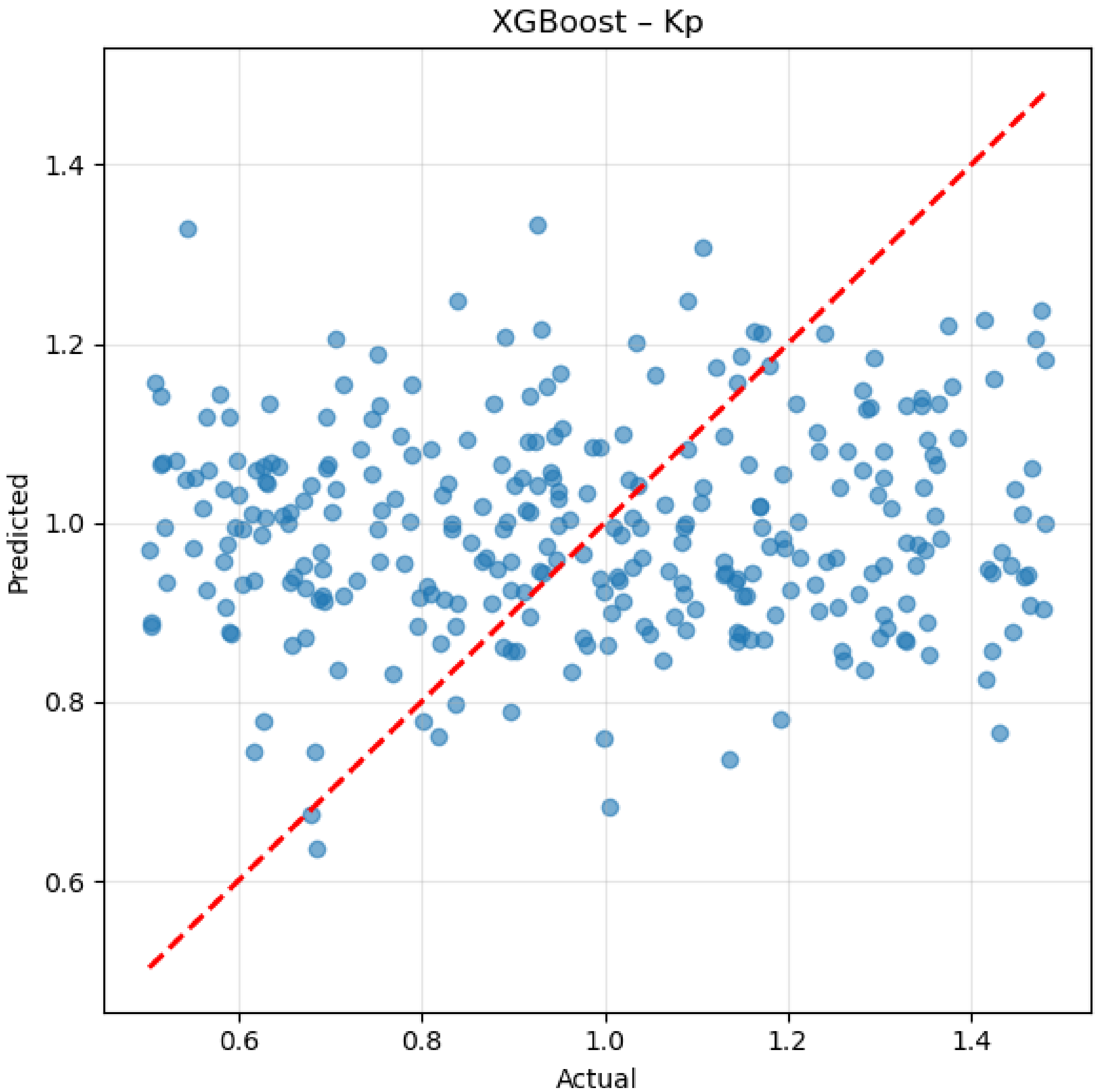
[49]:
```
for target, models in regression_predictions.items():
    for model_name, (y_true, y_pred) in models.items():
        plot_predicted_vs_actual(y_true, y_pred, target, model_name, OUT_DIR)
```



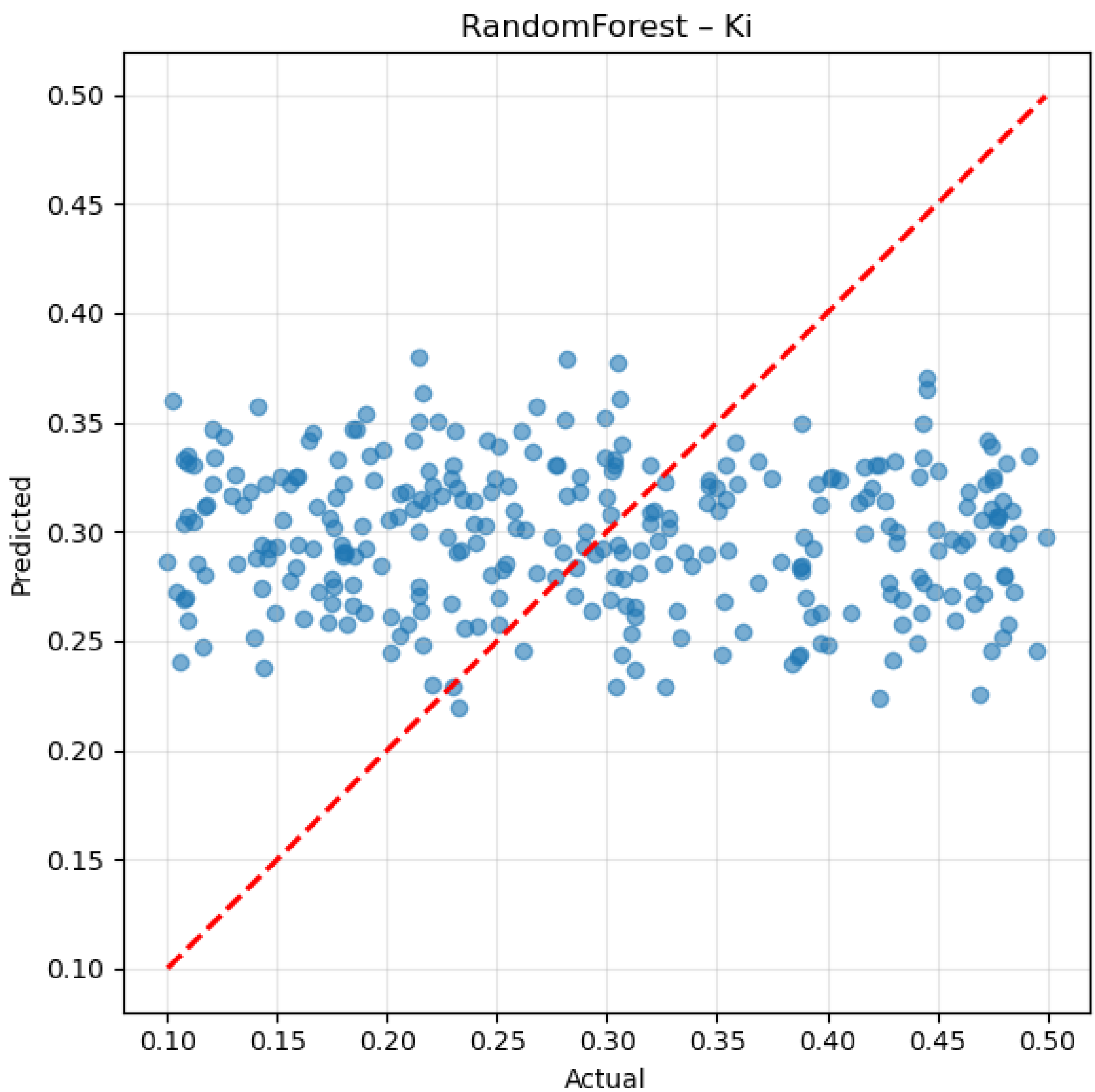

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\Pred_vs_Actual_Kp_RandomForest.png
```



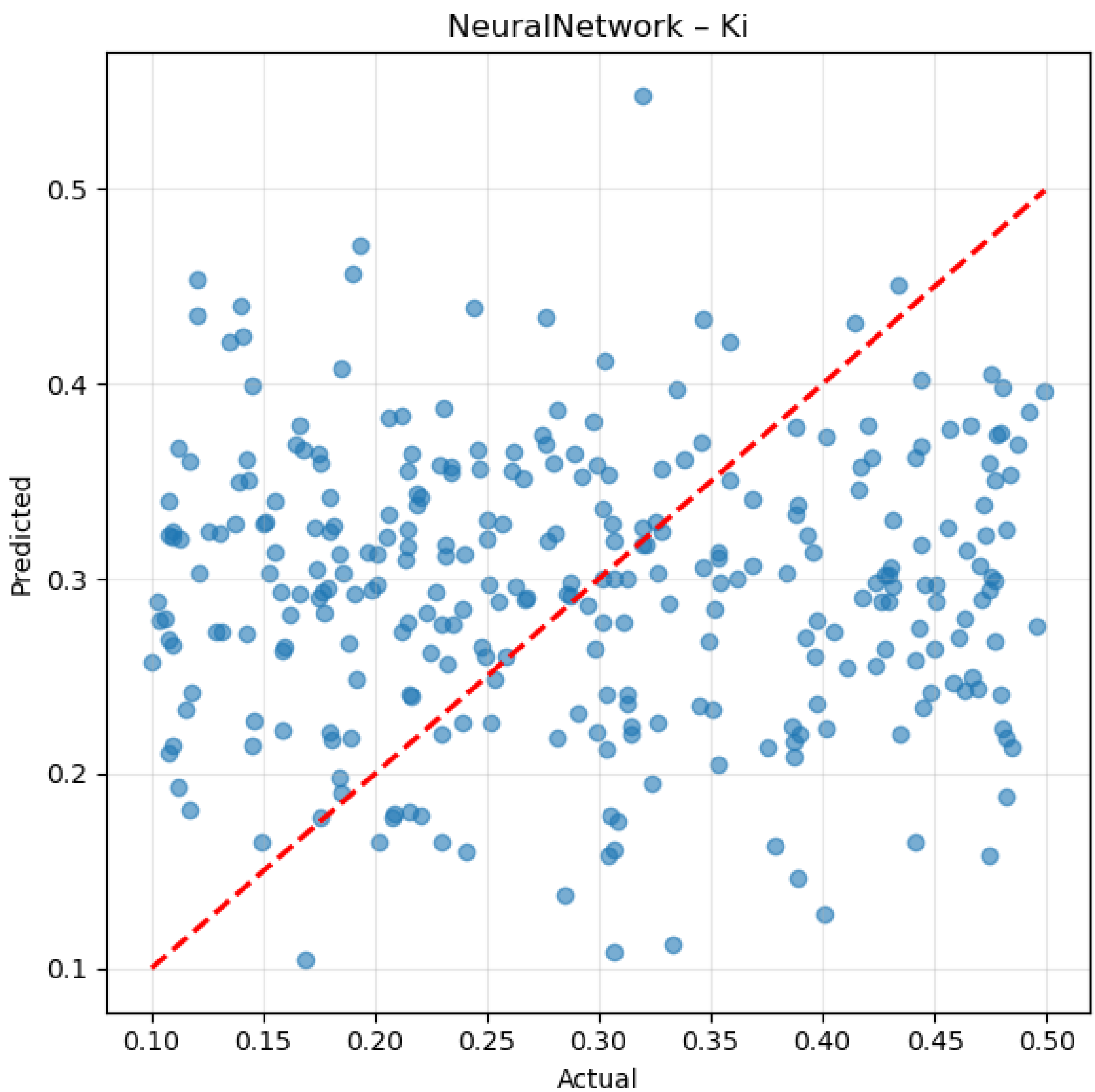
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Kp_NeuralNetwork.png



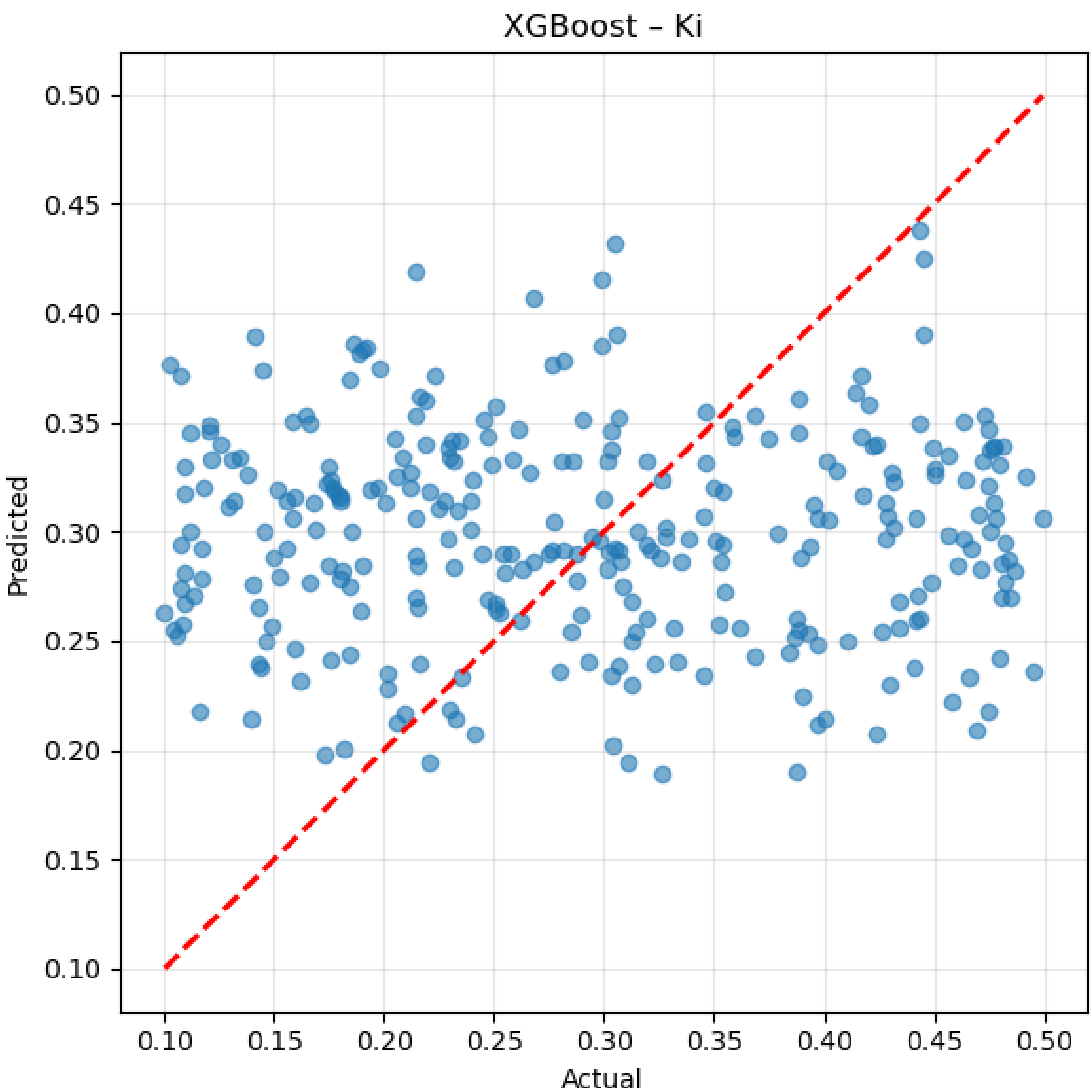

```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Kp_XGBoost.png
```



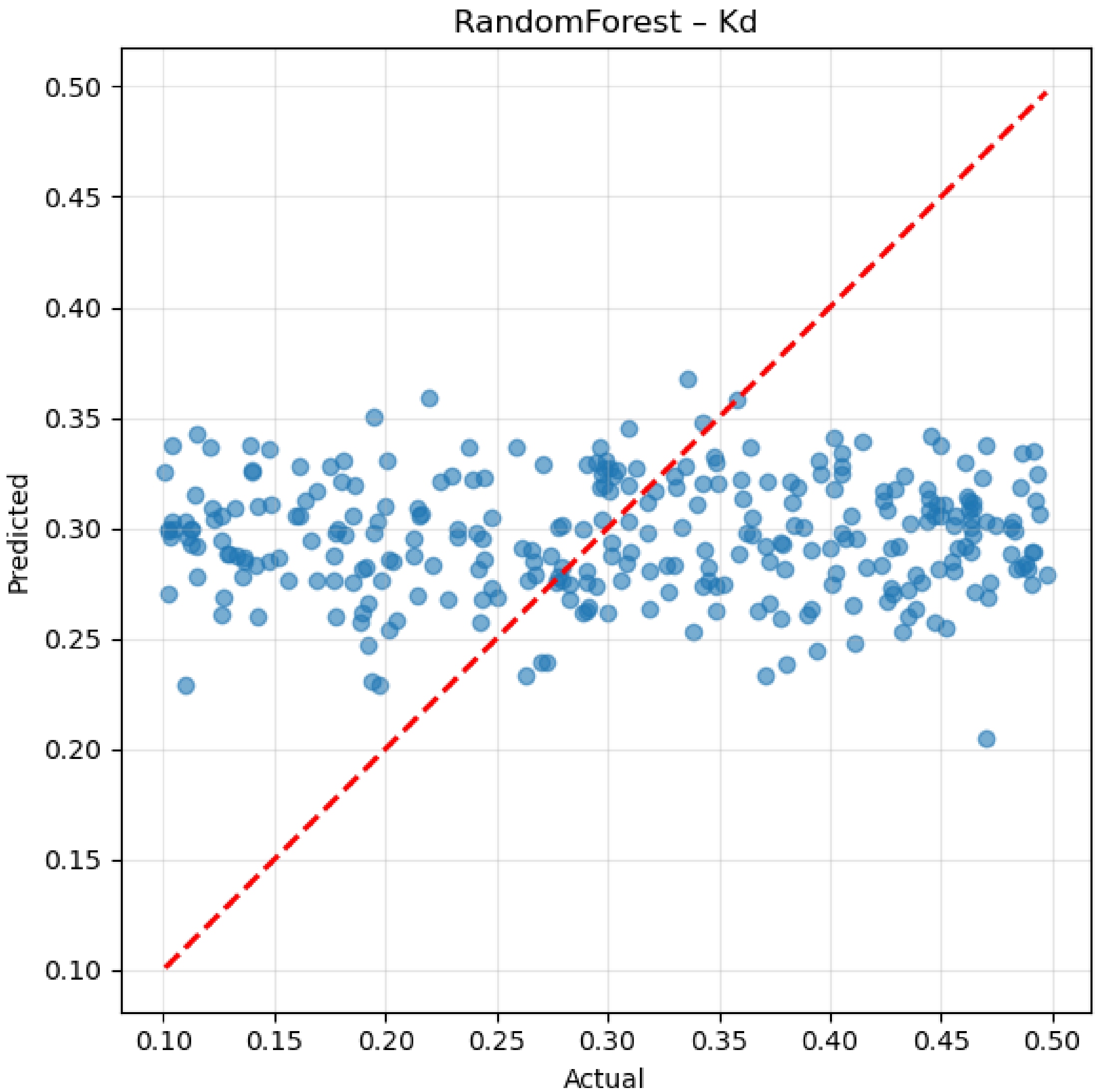
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Ki_RandomForest.png



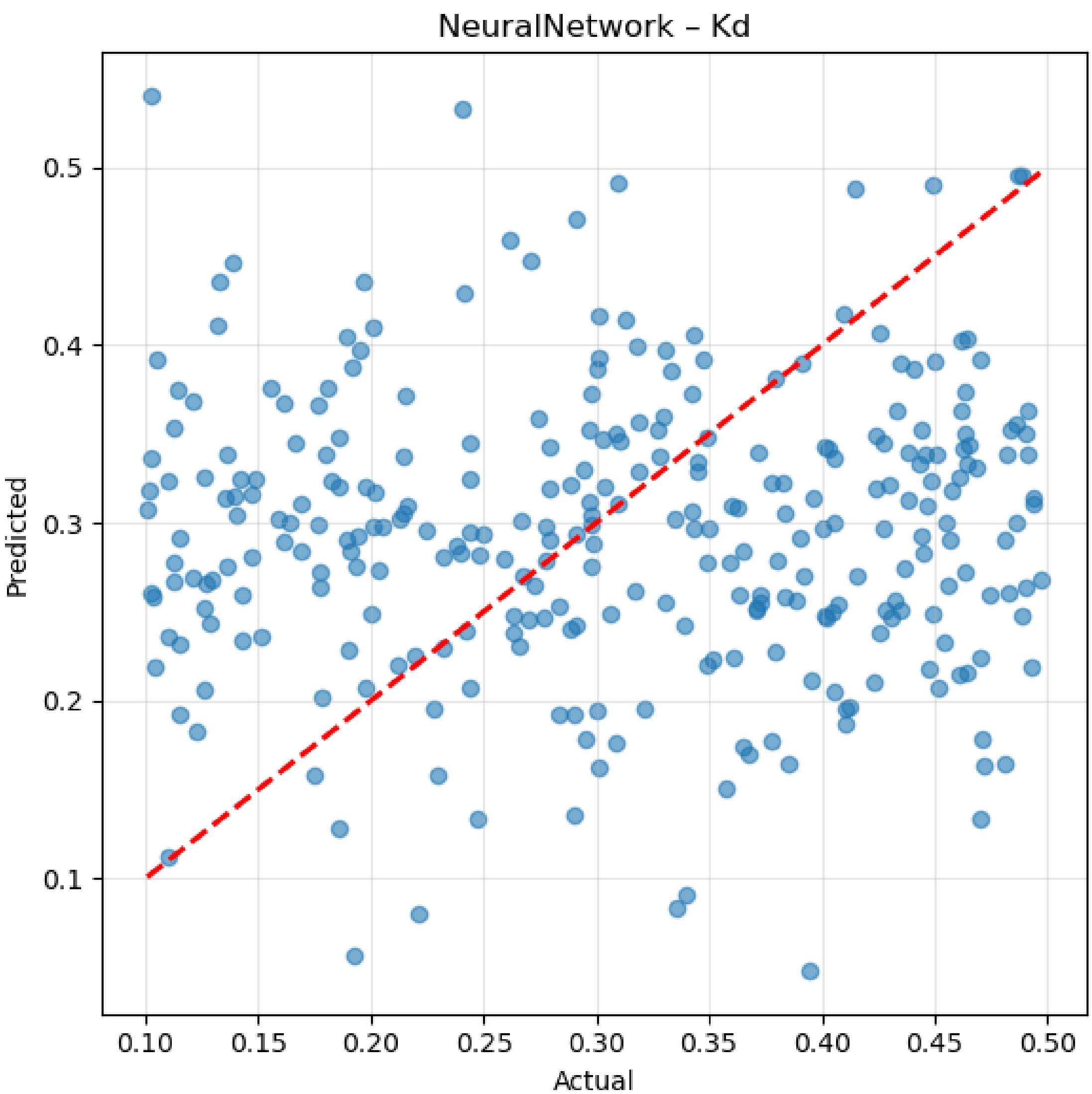
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Ki_NeuralNetwork.png



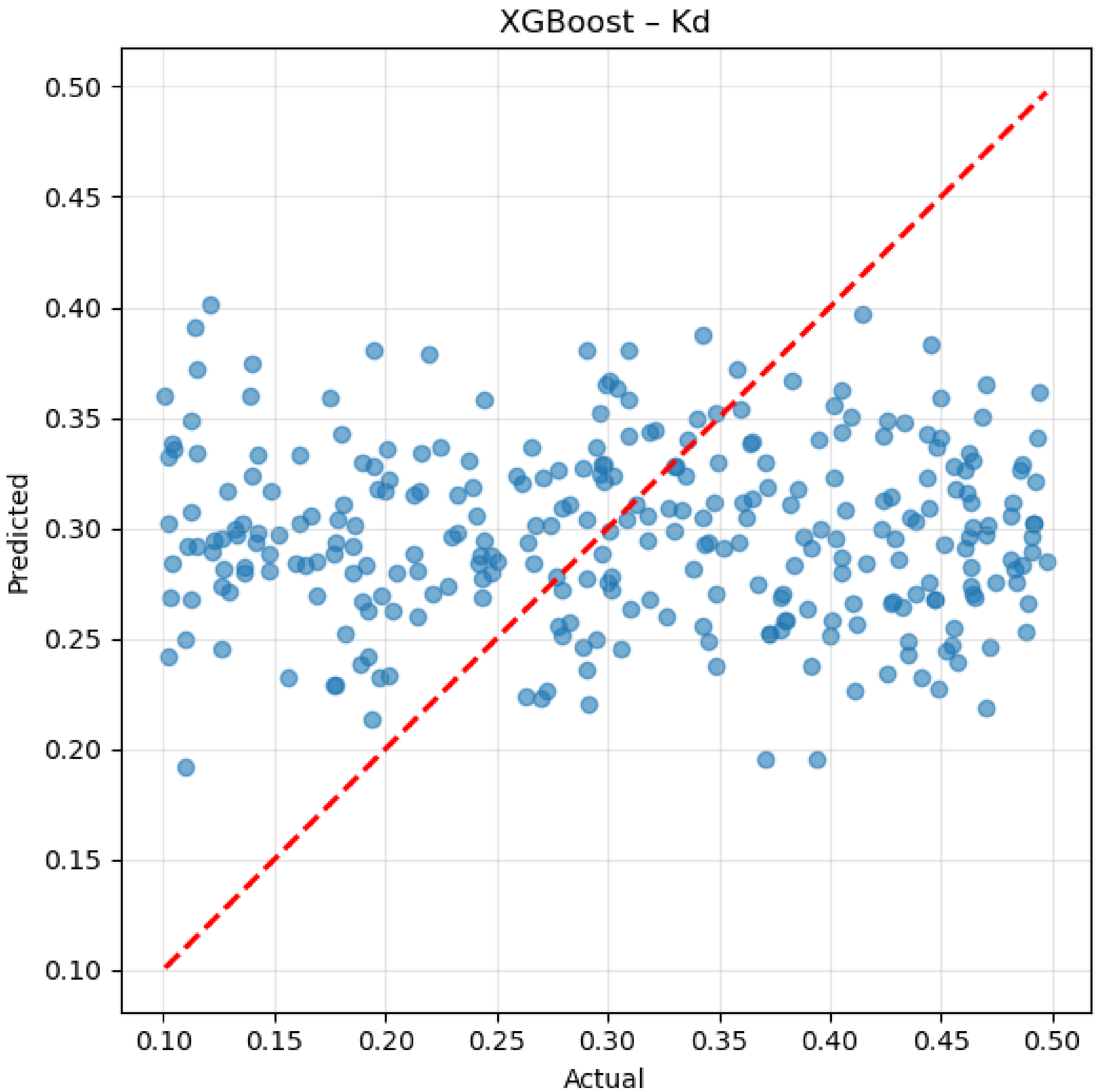
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Ki_XGBoost.png



Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Kd_RandomForest.png




```
Saved: I:\Self Study\python study\A Practical Industrial ML Applications for
Smart Manufacturing Temperature
Regulation\outputs\Pred_vs_Actual_Kd_NeuralNetwork.png
```



Saved: I:\Self Study\python study\A Practical Industrial ML Applications for Smart Manufacturing Temperature Regulation\outputs\Pred_vs_Actual_Kd_XGBoost.png

[54]:
```
ranking_regression = (
    regression_df
    .sort_values(["Target", "RMSE"])
    .assign(Rank=lambda df: df.groupby("Target")["RMSE"].rank())
)

rank_path = os.path.join(OUT_DIR, "PID_Gain_Regression_Ranking.csv")
ranking_regression.to_csv(rank_path, index=False)

print("Saved regression ranking →", rank_path)
```

```
display(ranking_regression)
```

```
Saved regression ranking → I:\Self Study\python study\A Practical Industrial ML
Applications for Smart Manufacturing Temperature
Regulation\outputs\PID_Gain_Regression_Ranking.csv
```

	Target	Model	RMSE	MAE	R2	Rank
6	Kd	RandomForest	0.121761	0.104089	-0.052561	1.0
8	Kd	XGBoost	0.126210	0.107421	-0.130882	2.0
7	Kd	NeuralNetwork	0.142427	0.118991	-0.440172	3.0
3	Ki	RandomForest	0.123399	0.106606	-0.122640	1.0
5	Ki	XGBoost	0.127433	0.109119	-0.197237	2.0
4	Ki	NeuralNetwork	0.137051	0.115517	-0.384777	3.0
0	Kp	RandomForest	0.282997	0.241050	-0.063437	1.0
2	Kp	XGBoost	0.294661	0.248000	-0.152906	2.0
1	Kp	NeuralNetwork	0.341680	0.280257	-0.550193	3.0

Note: The regression results for predicting continuous PID gains provide important insight into the nature of the dataset and the feasibility of direct gain prediction from operational variables. Across all three gains, Random Forest consistently achieved the lowest RMSE and MAE compared to Neural Networks and XGBoost, indicating relatively better numerical accuracy. However, the negative R2 values observed for all models and targets show that none of the regression models outperform a simple baseline that predicts the mean gain value. This suggests that the instantaneous process variables (temperature, error, ambient conditions, and control outputs) do not strongly explain the exact numerical tuning of PID gains present in the historical data.

These findings are consistent with typical industrial control practices, where PID gains are often fixed for long operating periods or adjusted manually in discrete steps rather than continuously optimized. As a result, many different operating states map to the same gain values, weakening the causal relationship that regression models rely on. The poorer performance of Neural Networks and XGBoost further indicates that increasing model complexity does not compensate for the lack of informative tuning signals in the available features, and may instead lead to overfitting or unstable generalization.

Overall, the results highlight a key limitation of direct PID gain regression from plant data while simultaneously validating the earlier classification-based approach. Coarse gain classification (e.g., low/medium/high) aligns better with how gains are selected in practice and provides more robust learning signals. These observations motivate alternative strategies such as gain scheduling, prediction of gain adjustments rather than absolute values, or reinforcement learning using a digital twin, which are more suitable for adaptive and intelligent temperature control systems.

0.0.2 To Compute the controller performance metrics

[61]:
```
# To find Overshoot, Settling time, Rise time, Steady-state error, IAE
↪(Integral Absolute Error) and ITAE (Integral Time Absolute Error)
```

[455]:
```
# 1. Create the Kp, Ki, Kd classes (Must match the training script's q=3
↪categorization)
df["Kp_class"] = pd.qcut(df["PID Kp"], q=3, labels=["Low", "Medium", "High"])
```

```python
df["Ki_class"] = pd.qcut(df["PID Ki"], q=3, labels=["Low", "Medium", "High"])
df["Kd_class"] = pd.qcut(df["PID Kd"], q=3, labels=["Low", "Medium", "High"])

# 2. Calculate the mean numeric value for each class (the Lookup Table)
Kp_lookup = df.groupby("Kp_class")["PID Kp"].mean().reset_index().
 ↪rename(columns={'PID Kp': 'Numeric Kp Value'})
Ki_lookup = df.groupby("Ki_class")["PID Ki"].mean().reset_index().
 ↪rename(columns={'PID Ki': 'Numeric Ki Value'})
Kd_lookup = df.groupby("Kd_class")["PID Kd"].mean().reset_index().
 ↪rename(columns={'PID Kd': 'Numeric Kd Value'})

# Dictionary structure is best for real-time lookups
GAIN_LOOKUP_TABLE = {
    'Kp': Kp_lookup.set_index("Kp_class")['Numeric Kp Value'].to_dict(),
    'Ki': Ki_lookup.set_index("Ki_class")['Numeric Ki Value'].to_dict(),
    'Kd': Kd_lookup.set_index("Kd_class")['Numeric Kd Value'].to_dict()
}

# Step 3: Display Output Tables
# ============================================================
print("\n--- Output Table 1: Kp Gain Lookup Table (ML Autotuning Mapping) ---")
print(Kp_lookup.to_markdown(index=False, numalign="left", stralign="left"))

print("\n--- Output Table 2: Ki Gain Lookup Table (ML Autotuning Mapping) ---")
print(Ki_lookup.to_markdown(index=False, numalign="left", stralign="left"))

print("\n--- Output Table 3: Kd Gain Lookup Table (ML Autotuning Mapping) ---")
print(Kd_lookup.to_markdown(index=False, numalign="left", stralign="left"))
```

```
--- Output Table 1: Kp Gain Lookup Table (ML Autotuning Mapping) ---
| Kp_class    | Numeric Kp Value    |
|:-----------|:-------------------|
| Low         | 0.663836            |
| Medium      | 0.981552            |
| High        | 1.3134              |

--- Output Table 2: Ki Gain Lookup Table (ML Autotuning Mapping) ---
| Ki_class    | Numeric Ki Value    |
|:-----------|:-------------------|
| Low         | 0.16952             |
| Medium      | 0.296566            |
| High        | 0.429568            |

--- Output Table 3: Kd Gain Lookup Table (ML Autotuning Mapping) ---
| Kd_class    | Numeric Kd Value    |
|:-----------|:-------------------|
```

```
| Low          | 0.169179           |
| Medium       | 0.303518           |
| High         | 0.434292           |
```

[63]:
```python
import joblib # Required to load saved models and scalers

# Placeholder for the Lookup Table calculated in Step 1
# In a real system, this dictionary would be loaded or hardcoded.
GAIN_LOOKUP_TABLE = {
    'Kp': {'Low': 0.6643, 'Medium': 0.9825, 'High': 1.3136},
    'Ki': {'Low': 0.1697, 'Medium': 0.2970, 'High': 0.4299},
    'Kd': {'Low': 0.1694, 'Medium': 0.3035, 'High': 0.4343}
}

# The corrected features used during training (must be consistent)
FEATURE_COLS = [
    "Current Temperature (°C)", "Setpoint Temperature (°C)", "Temperature Error␣
↪(°C)",
    "Ambient Temperature (°C)", "Humidity (%)", "Fuzzy Rule Base Parameters",
    "Temperature Error_lag1", "Temperature_delta", "Ambient_Temp_delta"
]

# Assuming models and scaler were saved after optimization:
# Kp_MODEL = joblib.load('optimized_kp_classifier.joblib')
# SCALER = joblib.load('feature_scaler.joblib')

# --- Conceptual Autotuning Function ---

def autotune_pid_gains_realtime(sensor_data, ML_models, SCALER, LOOKUP_TABLE,␣
↪FEATURE_COLS):
    """
    Executes the ML-assisted Gain Scheduling prediction and mapping.

    Args:
        sensor_data (dict): Dictionary of current sensor data (e.g., from a␣
↪PLC).
        ML_models (dict): Dictionary containing the trained Kp, Ki, Kd␣
↪classifiers.
        SCALER (StandardScaler): The fitted scaler object.
        LOOKUP_TABLE (dict): The numeric mapping of classes to values.
        FEATURE_COLS (list): The list of features required by the model.
    """
    # 1. Data Preparation and Feature Engineering
    input_df = pd.DataFrame([sensor_data])

    # NOTE: In a live system, the lag and delta features must be calculated
    # using the current sensor reading and the previous reading.
```

```
    # 2. Scaling
    input_X = input_df[FEATURE_COLS]
    input_X_scaled = SCALER.transform(input_X)

    # 3. ML Inference (Prediction)
    pred_kp_class = ML_models['Kp'].predict(input_X_scaled)[0]
    pred_ki_class = ML_models['Ki'].predict(input_X_scaled)[0]
    pred_kd_class = ML_models['Kd'].predict(input_X_scaled)[0]

    # 4. Mapping (The Autotuning Step)
    new_Kp = LOOKUP_TABLE['Kp'][pred_kp_class]
    new_Ki = LOOKUP_TABLE['Ki'][pred_ki_class]
    new_Kd = LOOKUP_TABLE['Kd'][pred_kd_class]

    recommended_gains = {
        'Kp': new_Kp,
        'Ki': new_Ki,
        'Kd': new_Kd,
        'Kp_Class': pred_kp_class,
        'Ki_Class': pred_ki_class,
        'Kd_Class': pred_kd_class
    }

    print("\n--- ML-Assisted Autotuning Result ---")
    print(f"Predicted Classes: Kp={pred_kp_class}, Ki={pred_ki_class},␣
↪Kd={pred_kd_class}")
    print(f"Recommended Numeric Gains: Kp={new_Kp:.4f}, Ki={new_Ki:.4f},␣
↪Kd={new_Kd:.4f}")

    # 5. Output to Controller (e.g., write_to_plc('Kp_REGISTER', new_Kp))
    return recommended_gains
```

[67]:

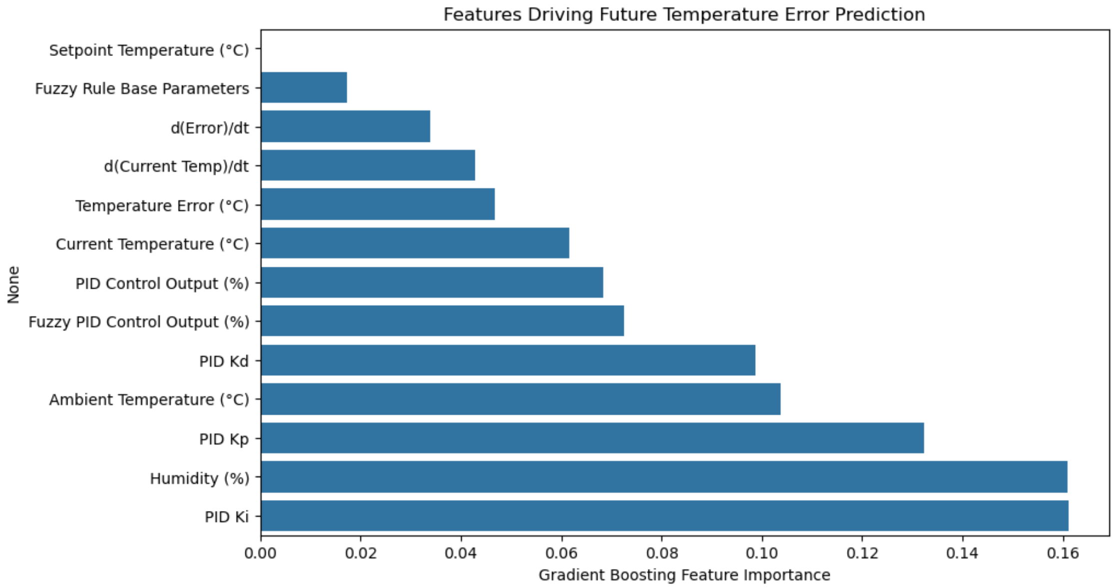
```
# --- Configuration ---
OUT_DIR = "outputs"
os.makedirs(OUT_DIR, exist_ok=True)
TARGET_PREDICT = 'Future_Temperature_Error'

# Load the dataset
try:
    df = pd.read_csv(DATA_PATH)
except FileNotFoundError:
    print(f"Error: Dataset file not found at {DATA_PATH}")
    raise

# --- 1. Dynamic Feature Engineering and Target Creation (FIXED) ---
```

```
# Convert Timestamp to datetime and sort (crucial for time series prediction)
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df = df.sort_values('Timestamp').reset_index(drop=True)

# Calculate the time difference (delta t) in seconds
# Handle the first row NaN by filling it with the mean of the rest of the time␣
↪deltas
time_diff = df['Timestamp'].diff().dt.total_seconds()
df['Time_Delta_s'] = time_diff.fillna(time_diff.mean())

# Calculate rate of change (derivative) of key variables
# The .diff() introduces a NaN in the first row, which is handled in the final␣
↪df.dropna()
df['d(Error)/dt'] = df['Temperature Error (°C)'].diff() / df['Time_Delta_s']
df['d(Current Temp)/dt'] = df['Current Temperature (°C)'].diff() /␣
↪df['Time_Delta_s']

# Create the Target Variable: Error at the next time step (Error_t+1)
# Shift the current error backward by 1. The last row will become NaN.
df[TARGET_PREDICT] = df['Temperature Error (°C)'].shift(-1)

# --- 2. Define Features and Target (No change here, as the columns are now␣
↪present) ---

# Features available at time 't' that influence error at 't+1'
features = [
    'Current Temperature (°C)',
    'Setpoint Temperature (°C)',
    'Temperature Error (°C)', # Error at time 't'
    'Ambient Temperature (°C)',
    'Humidity (%)',
    'PID Control Output (%)',
    'Fuzzy PID Control Output (%)',
    'PID Kp',
    'PID Ki',
    'PID Kd',
    'Fuzzy Rule Base Parameters',
    'd(Error)/dt',          # Dynamic feature
    'd(Current Temp)/dt'    # Dynamic feature
]
target = TARGET_PREDICT

# Drop ALL remaining NaN values (first row for diff, last row for shift(-1))
df_clean = df.dropna().reset_index(drop=True)

X = df_clean[features]
Y = df_clean[target]
```

```python
# --- 3. Model Training (Gradient Boosting Regressor) ---

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
↪random_state=42)

model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,␣
↪max_depth=3, random_state=42)

print(f"Training Gradient Boosting Regressor to predict {TARGET_PREDICT}␣
↪(Error_t+1)...")
model.fit(X_train, Y_train)
print("Training complete.")

# --- 4. Evaluation ---

Y_pred = model.predict(X_test)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print("\n--- Model Performance for Future Temperature Error Prediction (RE-RUN)␣
↪---")
print(f"  R-squared (R2): {r2:.4f}")
print(f"  Mean Absolute Error (MAE): {mae:.4f}")

# --- 5. Feature Importance Analysis ---
importance = model.feature_importances_
feature_names = X.columns
sorted_idx = importance.argsort()

# Create feature importance plot
plt.figure(figsize=(10, 6))
sns.barplot(x=importance[sorted_idx], y=feature_names[sorted_idx])
plt.xlabel("Gradient Boosting Feature Importance")
plt.title(f"Features Driving Future Temperature Error Prediction")
importance_plot_filename = os.path.join(OUT_DIR,␣
↪f"future_error_importance_rerun.png")
plt.savefig(importance_plot_filename)
plt.show()

print(f"\nFeature Importance Plot saved to: {importance_plot_filename}")
```

```
Training Gradient Boosting Regressor to predict Future_Temperature_Error
(Error_t+1)…
Training complete.

--- Model Performance for Future Temperature Error Prediction (RE-RUN) ---
```

```
R-squared (R2): -0.1073
Mean Absolute Error (MAE): 13.0284
```




```
Feature Importance Plot saved to: outputs\future_error_importance_rerun.png
```

[471]:

```python
import pandas as pd
import numpy as np
import os

# --- Configuration ---
# DATA_PATH = "Smart Manufacturing Temperature Regulation Dataset.csv"
SP_BAND_PERCENT = 0.02 # 2% band for Settling Time

# Load the dataset
df = pd.read_csv(DATA_PATH)

# --- 1. Preprocessing and Run Identification ---

# Convert Timestamp and sort data
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df = df.sort_values('Timestamp').reset_index(drop=True)

# Identify runs based on changes in Setpoint Temperature
df['Run_ID'] = (df['Setpoint Temperature (°C)'] != df['Setpoint Temperature␣
↪(°C)'].shift(1)).cumsum()
```

```python
# --- 2. Define Metrics Calculation Function (Essential for generating the␣
↪metrics) ---

def calculate_metrics_for_run(run_df, sp_band_percent):
    """Calculates control performance metrics for a single, continuous control␣
↪run."""
    if run_df.empty:
        return pd.Series()

    SP = run_df['Setpoint Temperature (°C)'].iloc[0]
    T_start = run_df['Current Temperature (°C)'].iloc[0]
    T_step = SP - T_start

    # Time relative to the start of the run
    run_df['Time'] = (run_df['Timestamp'] - run_df['Timestamp'].min()).dt.
↪total_seconds()

    # --- Steady-State Error (SSE) & Overshoot ---
    final_time = run_df['Time'].max()
    final_period_df = run_df[run_df['Time'] >= final_time * 0.9]
    sse = final_period_df['Temperature Error (°C)'].abs().mean()
    overshoot_abs = run_df['Current Temperature (°C)'].max() - SP
    overshoot = max(0, overshoot_abs)

    # --- Rise Time (10% to 90%) ---
    T_10 = T_start + 0.10 * T_step
    T_90 = T_start + 0.90 * T_step
    time_10 = run_df[run_df['Current Temperature (°C)'] >= T_10]['Time'].min()
    time_90 = run_df[run_df['Current Temperature (°C)'] >= T_90]['Time'].min()
    rise_time = time_90 - time_10 if pd.notna(time_10) and pd.notna(time_90)␣
↪and time_90 > time_10 else np.nan

    # --- Settling Time (within 2% band of SP) ---
    sp_band = SP * sp_band_percent
    outside_band = run_df[run_df['Temperature Error (°C)'].abs() > sp_band]

    if outside_band.empty:
        settling_time = 0.0
    else:
        last_outside_time = outside_band['Time'].max()
        last_error = np.abs(run_df['Current Temperature (°C)'].iloc[-1] - SP)
        settling_time = last_outside_time if last_error <= sp_band else np.nan

    # --- IAE and ITAE (Integral Metrics) ---
    run_df['dt_run'] = run_df['Time'].diff().fillna(0)
    iae = (run_df['Temperature Error (°C)'].abs() * run_df['dt_run']).sum()
```

```
    itae = (run_df['Temperature Error (°C)'].abs() * run_df['Time'] *␣
↪run_df['dt_run']).sum()

    return pd.Series({
        'Computed Overshoot (°C)': overshoot,
        'Computed Steady-State Error (°C)': sse,
        'Computed Rise Time (s)': rise_time,
        'Computed Settling Time (s)': settling_time,
        'Computed IAE': iae,
        'Computed ITAE': itae
    })

# --- 3. Apply Function and Aggregate Results ---

# Group by Run_ID and apply the calculation function to get metrics per run
run_metrics = df.groupby('Run_ID').apply(
    lambda x: calculate_metrics_for_run(x, SP_BAND_PERCENT)
).reset_index()


# --- 4. Calculate and Display Summary Statistics ---

# Select the computed columns and aggregate for mean, std, min, max
summary_stats = run_metrics[[
    'Computed Overshoot (°C)',
    'Computed Steady-State Error (°C)',
    'Computed Rise Time (s)',
    'Computed Settling Time (s)',
    'Computed IAE',
    'Computed ITAE'
]].agg(['mean', 'std', 'min', 'max']).T

print("\n--- Summary Statistics of Computed Performance Metrics (Across All␣
↪Runs) ---")
print(summary_stats.to_markdown(numalign="left", stralign="left", floatfmt=".
↪4f"))
```

```
--- Summary Statistics of Computed Performance Metrics (Across All Runs) ---
|                                  | mean                | std    | min
| max                 |
|:---------------------------------|:--------------------|:-------|:--------------
-----|:--------------------|
| Computed Overshoot (°C)          | 0.0000              | nan    | 0.0000
| 0.0000              |
| Computed Steady-State Error (°C) | 26.4405             | nan    | 26.4405
| 26.4405             |
```

```
| Computed Rise Time (s)          | nan               | nan   | nan
| nan               |
| Computed Settling Time (s)      | nan               | nan   | nan
| nan               |
| Computed IAE                    | 7636769.9438      | nan   | 7636769.9438
| 7636769.9438      |
| Computed ITAE                   | 1155067126435.4165 | nan   |
1155067126435.4165 | 1155067126435.4165 |
```

[ ]:

[ ]: