Bias-Variance Decomposition

In [56]:
```python
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

In [57]:
```python
# Import necessary libraries
import numpy as np
import pandas as pd

from sklearn.datasets import fetch_california_housing, load_iris, make_regre
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.metrics import mean_squared_error
from sklearn.utils import resample

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

Define Bias-Variance Decomposition Function

In [58]:
```python
def bias_variance_decomp(model, X_train, y_train, X_test, y_test, n_bootstra
    """Simple bias-variance decomposition for regression"""
    # Array to store predictions from each bootstrap model
    predictions = np.zeros((n_bootstraps, len(y_test)))

    # For each bootstrap sample
    for i in range(n_bootstraps):
        # Create bootstrap sample
        X_boot, y_boot = resample(X_train, y_train, random_state=i)

        # Train model on bootstrap sample
        model.fit(X_boot, y_boot)

        # Predict on test data
        predictions[i] = model.predict(X_test)

    # Average prediction across all bootstrap models
    average_pred = np.mean(predictions, axis=0)

    # Calculate squared bias
    bias_squared = np.mean((average_pred - y_test) ** 2)

    # Calculate variance
    variance = np.mean(np.var(predictions, axis=0))

    # Calculate average error
    error = np.mean(np.mean((predictions - y_test.reshape(1, -1)) ** 2, axis
```

```
    # Noise (irreducible error)
    noise = error - bias_squared - variance

    return bias_squared, variance, error, noise
```

1. California Housing Dataset

In [59]:
```
# Load California Housing dataset
california = fetch_california_housing()
X_california = california.data
y_california = california.target

# Scale features
scaler = StandardScaler()
X_california_scaled = scaler.fit_transform(X_california)

# Split data
X_train_cal, X_test_cal, y_train_cal, y_test_cal = train_test_split(
    X_california_scaled, y_california, test_size=0.2, random_state=42
)

print(f"California Housing Dataset: {X_california.shape}")
```

```
California Housing Dataset: (20640, 8)
```

In [60]:
```
# Define different complexity levels for decision trees
max_depths = [1, 3, 5, 10, None]

# Store results
bias_values_cal = []
variance_values_cal = []
error_values_cal = []

# Perform bias-variance decomposition
print("California Housing Dataset Results:")
print("-" * 60)
print(f"{'Max Depth':<10} {'Bias²':<15} {'Variance':<15} {'Total Error':<15}
print("-" * 60)

for depth in max_depths:
    model = DecisionTreeRegressor(max_depth=depth, random_state=42)
    bias, variance, error, noise = bias_variance_decomp(
        model, X_train_cal, y_train_cal, X_test_cal, y_test_cal
    )

    bias_values_cal.append(bias)
    variance_values_cal.append(variance)
    error_values_cal.append(error)

    depth_str = str(depth) if depth is not None else "None"
    print(f"{depth_str:<10} {bias:<15.4f} {variance:<15.4f} {error:<15.4f}")
print("-" * 60)
```

```
California Housing Dataset Results:
------------------------------------------------------------
Max Depth  Bias²           Variance        Total Error
------------------------------------------------------------
1          0.9177          0.0283          0.9460
1          0.9177          0.0283          0.9460
3          0.5990          0.0482          0.6472
3          0.5990          0.0482          0.6472
5          0.4615          0.0712          0.5327
5          0.4615          0.0712          0.5327
10         0.2972          0.1586          0.4558
10         0.2972          0.1586          0.4558
None       0.2548          0.2999          0.5547
------------------------------------------------------------
None       0.2548          0.2999          0.5547
------------------------------------------------------------
```

2. Iris Dataset

In [61]:
```python
# Define a simplified bias-variance decomposition for classification
def bias_variance_decomp_clf(model, X_train, y_train, X_test, y_test, n_boot
    """Simplified bias-variance estimation for classification"""
    predictions = np.zeros((n_bootstraps, len(y_test)))

    # For each bootstrap sample
    for i in range(n_bootstraps):
        # Create bootstrap sample
        X_boot, y_boot = resample(X_train, y_train, random_state=i)

        # Train model on bootstrap sample
        model.fit(X_boot, y_boot)

        # Predict on test data
        predictions[i] = model.predict(X_test)

    # Mode prediction (most common class) for each test point
    from scipy import stats
    main_predictions = stats.mode(predictions, axis=0, keepdims=False)[0]

    # Bias - error between main prediction and true class
    bias = np.mean(main_predictions != y_test)

    # Variance - disagreement between individual models
    variance = np.mean([np.mean(pred != main_predictions) for pred in predic

    # Total error - average misclassification rate
    error = np.mean([np.mean(pred != y_test) for pred in predictions])

    return bias, variance, error, 0  # Noise is 0 for this simplified approa
```

In [62]:
```python
# Load Iris dataset
iris = load_iris()
X_iris = iris.data
y_iris = iris.target
```

```python
# Scale features
scaler = StandardScaler()
X_iris_scaled = scaler.fit_transform(X_iris)

# Split data
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(
    X_iris_scaled, y_iris, test_size=0.2, random_state=42, stratify=y_iris
)

print(f"Iris Dataset: {X_iris.shape}")
```

```
Iris Dataset: (150, 4)
```

In [63]:

```python
# Define different complexity levels for decision trees
max_depths_iris = [1, 2, 3, 5, None]

# Store results
bias_values_iris = []
variance_values_iris = []
error_values_iris = []

# Perform bias-variance decomposition
print("Iris Dataset Results:")
print("-" * 60)
print(f"{'Max Depth':<10} {'Bias':<15} {'Variance':<15} {'Total Error':<15}"
print("-" * 60)

for depth in max_depths_iris:
    model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    bias, variance, error, _ = bias_variance_decomp_clf(
        model, X_train_iris, y_train_iris, X_test_iris, y_test_iris
    )

    bias_values_iris.append(bias)
    variance_values_iris.append(variance)
    error_values_iris.append(error)

    depth_str = str(depth) if depth is not None else "None"
    print(f"{depth_str:<10} {bias:<15.4f} {variance:<15.4f} {error:<15.4f}")
print("-" * 60)
```

```
Iris Dataset Results:
------------------------------------------------------------
Max Depth  Bias           Variance       Total Error
------------------------------------------------------------
1          0.1000         0.3367         0.3440
2          0.0667         0.0347         0.0720
3          0.0333         0.0287         0.0620
3          0.0333         0.0287         0.0620
5          0.0667         0.0367         0.0727
None       0.0667         0.0367         0.0727
------------------------------------------------------------
5          0.0667         0.0367         0.0727
None       0.0667         0.0367         0.0727
------------------------------------------------------------
```

3. Random Dataset

In [64]:
```
# Generate random dataset
X_random, y_random = make_regression(
    n_samples=500, n_features=10, n_informative=5, noise=0.3, random_state=4
)

# Split data
X_train_random, X_test_random, y_train_random, y_test_random = train_test_sp
    X_random, y_random, test_size=0.2, random_state=42
)

print(f"Random Dataset: {X_random.shape}")
```

```
Random Dataset: (500, 10)
```

In [65]:
```
# Define different complexity levels for decision trees
max_depths_random = [1, 3, 5, 10, None]

# Store results
bias_values_random = []
variance_values_random = []
error_values_random = []

# Perform bias-variance decomposition
print("Random Dataset Results:")
print("-" * 60)
print(f"{'Max Depth':<10} {'Bias²':<15} {'Variance':<15} {'Total Error':<15}
print("-" * 60)

for depth in max_depths_random:
    model = DecisionTreeRegressor(max_depth=depth, random_state=42)
    bias, variance, error, noise = bias_variance_decomp(
        model, X_train_random, y_train_random, X_test_random, y_test_random
    )

    bias_values_random.append(bias)
    variance_values_random.append(variance)
    error_values_random.append(error)

    depth_str = str(depth) if depth is not None else "None"
    print(f"{depth_str:<10} {bias:<15.4f} {variance:<15.4f} {error:<15.4f}")
print("-" * 60)
```

```
Random Dataset Results:
------------------------------------------------------------
Max Depth  Bias²           Variance        Total Error
------------------------------------------------------------
1          2530.0593       193.7232        2723.7825
1          2530.0593       193.7232        2723.7825
3          1381.7954       748.9352        2130.7307
3          1381.7954       748.9352        2130.7307
5          845.9926        991.4796        1837.4723
5          845.9926        991.4796        1837.4723
10         606.1889        1180.8849       1787.0738
10         606.1889        1180.8849       1787.0738
None       615.8563        1170.2882       1786.1445
------------------------------------------------------------
None       615.8563        1170.2882       1786.1445
------------------------------------------------------------
```