STAT312: Nonparametric Regression 03

Fakade, Ncumisa (225091410)

10 October 2025

Important Instructions

Code Completion Required: This practical contains code chunks with missing elements marked by `___` placeholders. You must replace these placeholders with the correct R code to complete each task. Additionally, many code chunks are set to `eval = FALSE` to prevent execution errors whilst code remains incomplete. Once you have filled in the missing code for a chunk, change the chunk option to `eval = TRUE` to enable execution and generate results.

Yellow Boxes

You do not need to write answers to questions in the yellow boxes. However, try to answer them for your own understanding.

Introduction

This practical covers **non-parametric regression methods**, which model relationships without assuming specific functional forms, offering flexibility unmatched by parametric models like linear regression.

Key Methods

- **k-Nearest Neighbours (k-NN) regression:** Predicts by averaging the response values of the k closest observations. A smaller k offers high flexibility/variance, while a larger k provides smoother, more stable predictions.
- **Nadaraya-Watson kernel regression:** An extension of k-NN that uses **kernel weighting** to give closer neighbours greater influence. The **bandwidth parameter** h controls the smoothness, similarly to how k controls it in k-NN.

These techniques are essential for capturing **non-linearity, interactions, and local data patterns**.

Objectives

By completing this practical, you will:

1. **Implement** k-NN and Nadaraya-Watson regression in R
2. **Understand** how tuning parameters (k and bandwidth h) affect model behaviour
3. **Apply** cross-validation for optimal parameter selection
4. **Visualise** fitted curves and decision boundaries
5. **Compare** non-parametric methods with parametric alternatives
6. **Interpret** results in the context of environmental science

Warning

This practical requires **independent problem-solving**. Code scaffolding is minimal. You must determine implementation details yourself. Budget approximately **120 minutes** for completion.

Required Packages

```
library(tidyverse)
library(caret)        # For k-NN regression
library(rsample)      # For cross-validation
library(broom)        # For model output formatting
library(knitr)        # For table formatting
library(gridExtra)    # For arranging plots
```

Environmental Context

Climate scientists study how atmospheric carbon dioxide (CO_2) concentrations relate to global temperature anomalies over the past century. They collected monthly observations measuring:

- **Temperature anomaly** (°C above historical baseline) — the response variable
- **CO_2 concentration** (parts per million) — the primary predictor

The relationship between CO_2 and temperature likely exhibits non-linear patterns due to complex atmospheric feedback mechanisms. Traditional linear models may fail to capture these dynamics adequately.

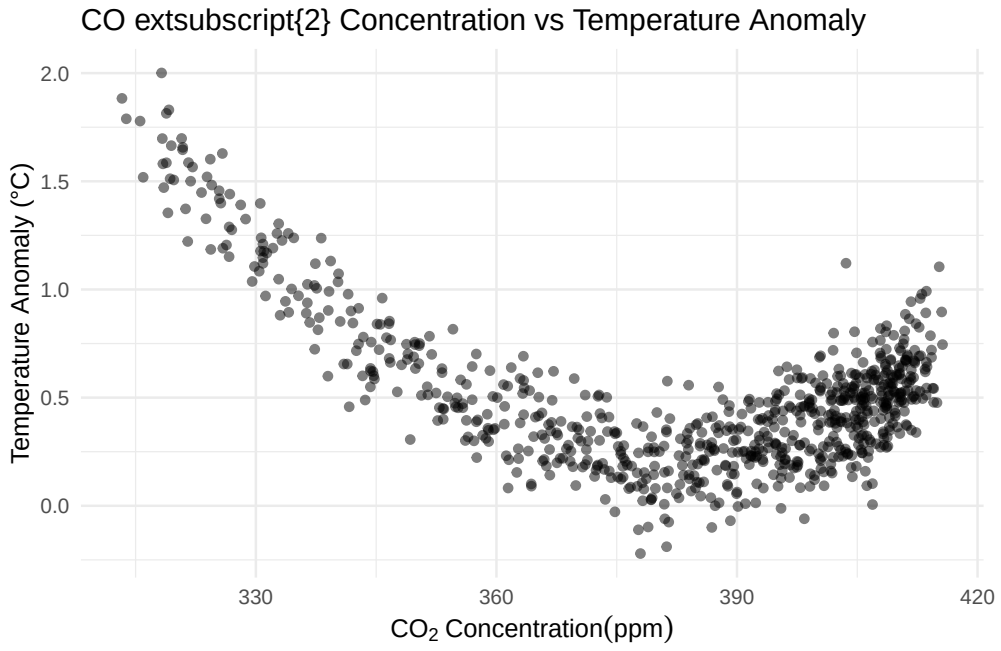
Task 0: Data Import and Visualisation

```
# Replace ___ with the correct filename in quotes
climate_data <- read_csv("climate_data.csv")

# Replace ___ with the function to examine data structure
glimpse(climate_data)
```

```
## Rows: 780
## Columns: 4
## $ year                 <dbl> 1960, 1960, 1960, 1960, 1960, 1960, 1960,
## $ month                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1,
## $ co2                  <dbl> 318.5175, 319.0331, 318.8432, 318.8379, 31
## $ temperature_anomaly <dbl> 1.470792, 1.354156, 1.585658, 1.814097, 1.8
```

The dataset contains *780* observations and *4* variables.

```
# Create scatter plot of the relationship
ggplot(climate_data, aes(x = co2, y = temperature_anomaly)) +
  geom_point(alpha = 0.5) +
  labs(
    title = "CO\textsubscript{2} Concentration vs Temperature Anomaly",
    x = expression(CO[2]~ Concentration (ppm)),
    y = "Temperature Anomaly (\u00B0C)",
  ) +
  theme_minimal()
```



```r
# Calculate correlation
# Replace ___ with the two variables to correlate
correlation <- cor(climate_data$co2, climate_data$temperature_anomaly)
cat(sprintf("Correlation: %.3f\n", correlation))
```

```
## Correlation: -0.534
```

Think about it

Does the relationship appear linear? What patterns do you observe? Would ordinary least squares regression be appropriate?

Task 1: Data Preparation and Train-Test Split

Before fitting models, you must prepare the data appropriately.

Your tasks:

1. **Create training and test sets** using 75-25 split with `initial_split()`
2. **Extract predictor and response as matrices/vectors** for both training and test sets

```r
# Create training and test sets (75-25 split)
set.seed(2024)
# Replace ___ with the function to create data splits
data_split <- initial_split(
  climate_data, prop = 0.75)
train_data <- training(data_split)
test_data <- testing(data_split)

cat(sprintf(
  "Training set: %d observations\n",
  nrow(train_data)))
```

```
## Training set: 585 observations
```

```
cat(sprintf(
  "Test set: %d observations\n",
  nrow(test_data)))
```

```
## Test set: 195 observations
```

```
# Extract predictors and response for training set
# Convert predictor to matrix format for knnreg()
x_train <- as.matrix(train_data$co2)
y_train <- train_data$temperature_anomaly

# Extract predictors and response for test set
x_test <- as.matrix(test_data$co2)
y_test <- test_data$temperature_anomaly

# Verify dimensions
cat(sprintf("\nx_train dimensions: %d x %d\n",
            nrow(x_train), ncol(x_train)))
```

```
##
## x_train dimensions: 585 x 1
```

```
cat(sprintf("y_train length: %d\n", length(y_train)))
```

```
## y_train length: 585
```

Task 2: k-Nearest Neighbours Regression

k-NN regression predicts each point by averaging the k nearest neighbours:

$$\hat{y}(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$$

where $\mathcal{N}_k(x)$ denotes the set of k nearest neighbours to point x.

Your tasks:

1. **Fit k-NN models** for multiple values of k
2. **Visualise** how k affects fitted curves
3. **Calculate** training and test errors for each k
4. **Identify** optimal k value

caret Package

The `caret::knnreg()` function requires:

- `x`: matrix or data frame of training predictors
- `y`: vector of training responses
- `k`: number of neighbours

The function returns a model object. Use `predict(model, newdata)` to generate predictions, where `newdata` must be a matrix with the same structure as `x`.

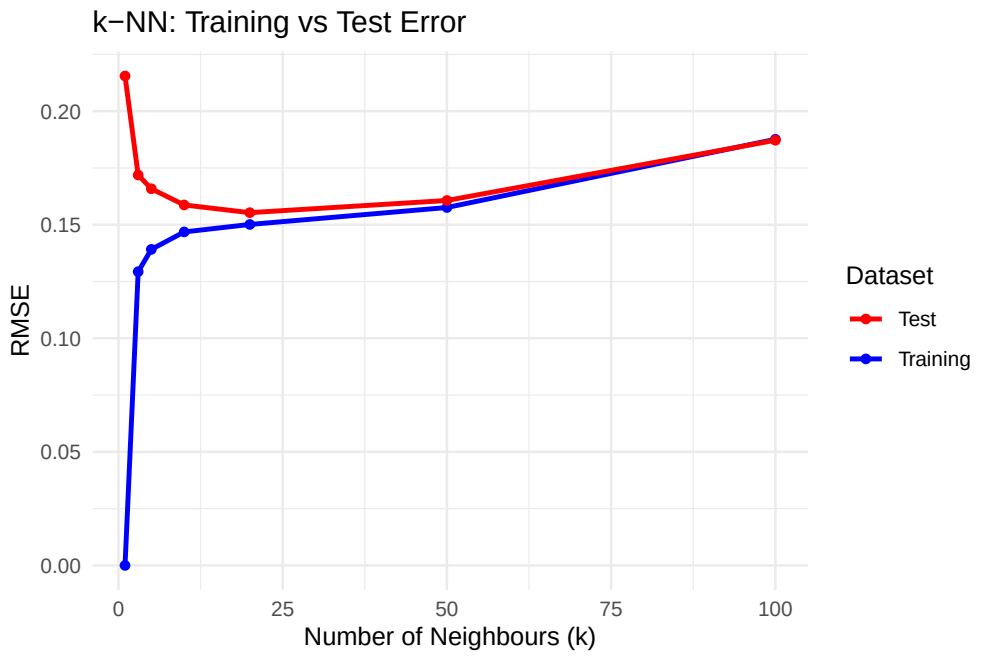
```
# Define range of k values to test
# We test a range from very flexible (k=1) to very smooth (k=100)
k_values <- c(1, 3, 5, 10, 20, 50, 100)

# Initialise storage for results
knn_results <- data.frame(
  k = integer(),
  train_rmse = numeric(),
  test_rmse = numeric()
)

# Fit k-NN models for each k value
for(k in k_values) {
  # Fit model on training data
```

```r
  # Replace ___ with appropriate arguments
  knn_model <- knnreg(
    x = x_train,
    y = y_train,
    k = k
  )

  # Predict training set
  # Replace ___ with appropriate arguments
  train_pred <- predict(knn_model, newdata = x_train)

  # Predict test set
  # Replace ___ with appropriate arguments
  test_pred <- predict(knn_model, newdata = x_test)

  # Calculate RMSE for training and test sets
  train_rmse <- sqrt(mean((train_pred - y_train)^2))
  test_rmse <- sqrt(mean((test_pred - y_test)^2))

  # Store results
  knn_results <- rbind(
    knn_results,
    data.frame(k = k, train_rmse = train_rmse, test_rmse = test_rmse)
  )
}

# Display results table
kable(knn_results, digits = 3,
      caption = "k-NN Performance for Different k Values")
```

Table 1: k-NN Performance for Different k Values

k	train_rmse	test_rmse
1	0.000	0.215
3	0.129	0.172
5	0.139	0.166
10	0.147	0.159
20	0.150	0.155
50	0.158	0.161
100	0.188	0.187

```
# Visualise how training and test error change with k
# Replace ___ to create appropriate plot
ggplot(knn_results, aes(x = k)) +
  geom_line(aes(y = train_rmse, colour = "Training"), size = 1) +
  geom_line(aes(y = test_rmse, colour = "Test"), size = 1) +
  geom_point(aes(y = train_rmse, colour = "Training")) +
  geom_point(aes(y = test_rmse, colour = "Test")) +
  scale_colour_manual(
    values = c("Training" = "blue", "Test" = "red")) +
  labs(
    title = "k-NN: Training vs Test Error",
    x = "Number of Neighbours (k)",
    y = "RMSE",
    colour = "Dataset"
  ) +
  theme_minimal()
```




```r
# Identify optimal k (minimum test RMSE)
# Replace ___ with appropriate code
optimal_k <- knn_results$k[which.min(knn_results$test_rmse)]
cat(sprintf("\nOptimal k: %d\n", optimal_k))
```

```
##
## Optimal k: 20
```

Consider

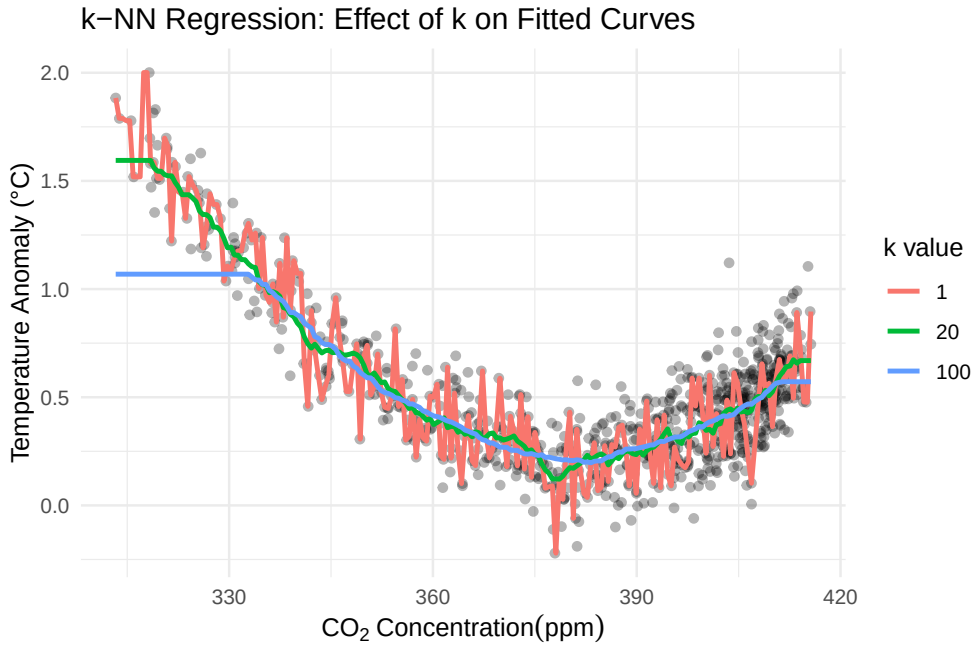
What happens to training and test error as k increases? Why does this occur? What does this reveal about the bias-variance trade-off?

Task 3: Visualising k-NN Predictions

```r
# Create grid of CO2 values for smooth prediction curves
co2_grid <- seq(
   min(climate_data$co2),
  max(climate_data$co2),
```

```
  length.out = 200)

# Convert to matrix for knnreg predictions
co2_grid_matrix <- as.matrix(co2_grid)

# Generate predictions for selected k values
k_selected <- c(1, 20, 100)
predictions_list <- list()

for(k in k_selected) {
  # Replace ___ with appropriate arguments
  knn_model <- knnreg(
    x = x_train,
    y = y_train,
    k = k
  )

  knn_pred <- predict(knn_model, newdata = co2_grid_matrix)

  predictions_list[[as.character(k)]] <- data.frame(
    co2 = co2_grid,
    prediction = knn_pred,
    k = k
  )
}

# Combine predictions
all_predictions <- bind_rows(predictions_list)

# Plot predictions with original data
```

```
ggplot() +
  geom_point(
    data = climate_data,
    aes(x = co2, y = temperature_anomaly),
    alpha = 0.3) +
  geom_line(
    data = all_predictions,
    aes(x = co2, y = prediction, colour = factor(k)),
    size = 1) +
  labs(
    title = "k-NN Regression: Effect of k on Fitted Curves",
    x = expression(CO[2]~ Concentration (ppm)),
    y = "Temperature Anomaly (\u00B0C)",
    colour = "k value"
  ) +
  theme_minimal()
```



Interpret

How does the smoothness of fitted curves change with k? Which k value appears most appropriate? What criteria would you use to decide?

Task 4: Cross-Validation for k Selection

Rather than relying on a single train-test split, you should use cross-validation for more robust parameter selection.

```
# Set up 10-fold cross-validation
set.seed(2024)
# Replace ___ with appropriate function and arguments
cv_folds <- vfold_cv(train_data, v = 10)

# Define range of k values
k_range <- c(1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 75, 100)

# Initialise storage
cv_results <- data.frame(
  k = integer(),
  fold = integer(),
  rmse = numeric()
)

# Perform cross-validation
for(k in k_range) {
  for(i in 1:nrow(cv_folds)) {
    # Extract fold data
    train_fold <- analysis(cv_folds$splits[[i]])
    test_fold <- assessment(cv_folds$splits[[i]])
```

```
    # Prepare matrices for this fold
    x_train_fold <- as.matrix(train_fold$co2)
    y_train_fold <- train_fold$temperature_anomaly
    x_test_fold <- as.matrix(test_fold$co2)

    # Fit k-NN model
    # Replace ___ with appropriate arguments
    knn_model <- knnreg(
      x = x_train_fold,
      y = y_train_fold,
      k = k
    )

    # Make predictions
    knn_pred <- predict(knn_model, newdata = x_test_fold)

    # Calculate RMSE
    fold_rmse <- sqrt(mean((test_fold$temperature_anomaly - knn_pred)^2))

    # Store results
    cv_results <- rbind(
      cv_results,
      data.frame(k = k, fold = i, rmse = fold_rmse)
    )
  }
}

# Calculate mean CV error for each k
# Replace ___ with appropriate dplyr code
```

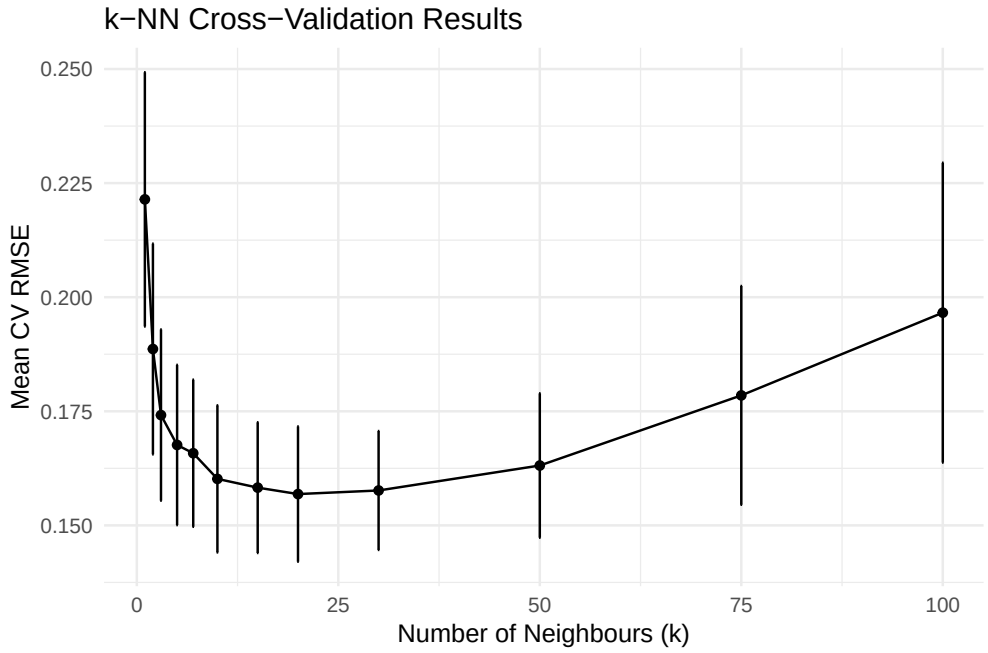
Table 2: Cross-Validation Results for k-NN

k	mean_rmse	sd_rmse
1	0.221	0.028
2	0.189	0.023
3	0.174	0.019
5	0.168	0.018
7	0.166	0.016
10	0.160	0.016
15	0.158	0.014
20	0.157	0.015
30	0.158	0.013
50	0.163	0.016
75	0.178	0.024
100	0.197	0.033

```
cv_summary <- cv_results %>%
  group_by(k) %>%
  summarise(
    mean_rmse = mean(rmse),
    sd_rmse = sd(rmse),
    .groups = 'drop'
  )

kable(cv_summary, digits = 3,
      caption = "Cross-Validation Results for k-NN")
```

```
# Plot CV results with error bars
# Replace ___ with appropriate ggplot code
ggplot(cv_summary, aes(x = k, y = mean_rmse)) +
  geom_line() +
  geom_point() +
  geom_errorbar(
```

```
    aes(
      ymin = mean_rmse - sd_rmse,
      ymax = mean_rmse + sd_rmse),
    width = 0.2) +
  labs(
    title = "k-NN Cross-Validation Results",
    x = "Number of Neighbours (k)",
    y = "Mean CV RMSE"
  ) +
  theme_minimal()
```




```
# Identify optimal k from CV
optimal_k_cv <- cv_summary$k[which.min(cv_summary$mean_rmse)]
cat(sprintf("\nOptimal k (CV): %d\n", optimal_k_cv))
```

```
##
## Optimal k (CV): 20
```

```
cat(sprintf("CV RMSE: %.4f\n",
            min(cv_summary$mean_rmse)))
```

```
## CV RMSE: 0.1569
```

Task 5: Implementing the Gaussian Kernel

Before implementing Nadaraya-Watson regression, you need to create a Gaussian kernel function. The Gaussian (or Normal) kernel is defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$

This kernel assigns weights to observations based on their distance from the prediction point. Closer observations receive higher weights.

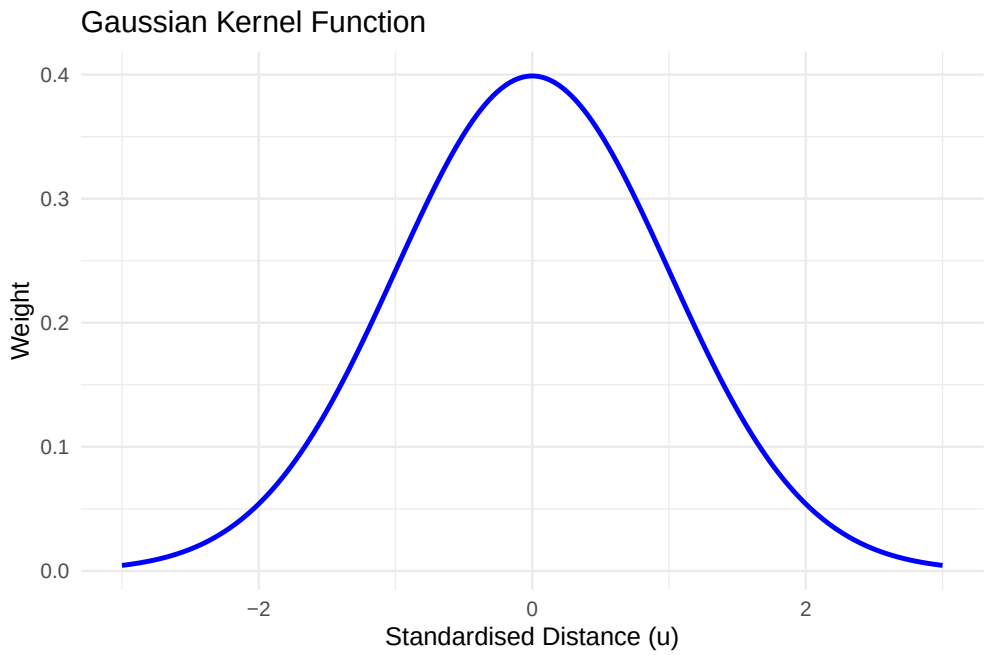
```
# Implement the Gaussian kernel function
# Replace ___ to complete the function
gaussian_kernel <- function(u) {
  # The Gaussian kernel formula
  # Hint: 2*pi in R is written as 2*pi
  (1 / sqrt(2*pi)) * exp(-(u^2) / 2)
}

# Test your kernel function
test_values <- c(-2, -1, 0, 1, 2)
kernel_weights <- gaussian_kernel(test_values)
cat("Kernel weights for test values:\n")
```

```
## Kernel weights for test values:
```

```r
print(data.frame(u = test_values, weight = kernel_weights))
```

```
##    u     weight
## 1 -2 0.05399097
## 2 -1 0.24197072
## 3  0 0.39894228
## 4  1 0.24197072
## 5  2 0.05399097
```

```r
# Visualise the kernel function
u_seq <- seq(-3, 3, length.out = 200)
kernel_plot_data <- data.frame(
  u = u_seq,
  weight = gaussian_kernel(u_seq)
)

ggplot(kernel_plot_data, aes(x = u, y = weight)) +
  geom_line(size = 1, colour = "blue") +
  labs(
    title = "Gaussian Kernel Function",
    x = "Standardised Distance (u)",
    y = "Weight"
  ) +
  theme_minimal()
```



Understanding the Kernel

What happens to the weight as the distance increases? Why does the Gaussian kernel produce smoother estimates than k-NN?

Task 6: Implementing Nadaraya-Watson Regression

The Nadaraya-Watson estimator is defined as:

$$\hat{y}(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) y_i}{\sum_{i=1}^{n} K_h(x - x_i)}$$

where $K_h(u) = \frac{1}{h} K\left(\frac{u}{h}\right)$ and h is the bandwidth parameter.

Your task: Implement a function that computes Nadaraya-Watson predictions.

```
# Implement Nadaraya-Watson estimator
# Replace ___ to complete the function
nadaraya_watson <- function(x_train, y_train, x_test, bandwidth) {
```

```r
  # Initialise vector to store predictions
  predictions <- numeric(length(x_test))

  # Loop through each test point
  for(i in 1:length(x_test)) {
    # Calculate distances from test point to all training points
    # Hint: x_test[i] is a single value, x_train is a vector
    distances <- x_test[i] - x_train

    # Calculate standardised distances (u = distance / bandwidth)
    u <- distances / bandwidth

    # Apply Gaussian kernel to get weights
    weights <- gaussian_kernel(u)

    # Calculate weighted average (Nadaraya-Watson formula)
    # Numerator: sum of weighted responses
    numerator <- sum(weights * y_train)

    # Denominator: sum of weights
    denominator <- sum(weights)

    # Compute prediction
    predictions[i] <- numerator / denominator
  }

  return(predictions)
}

# Test your function with a single bandwidth
```

```r
test_bandwidth <- 20
nw_test_pred <- nadaraya_watson(
  x_train = x_train,
  y_train = y_train,
  x_test = x_test,
  bandwidth = test_bandwidth
)

# Calculate test RMSE
nw_test_rmse <- sqrt(mean((nw_test_pred - y_test)^2))
cat(sprintf("Test RMSE (h = %.1f): %.4f\n", test_bandwidth, nw_test_rmse))
```

```
## Test RMSE (h = 20.0): 0.2191
```

Think about it

What happens if the denominator (sum of weights) is very small? How might you handle this edge case?

Task 7: Exploring Different Bandwidths

Now test your Nadaraya-Watson implementation across multiple bandwidth values.

```r
# Calculate Silverman's rule of thumb bandwidth
# h = 0.9 * min(sd, IQR/1.34) * n^(-1/5)
n <- length(x_train)
sigma <- sd(as.vector(x_train))
iqr <- IQR(as.vector(x_train))
h_silverman <- 0.9 * min(sigma, iqr / 1.34) * n^(-1/5)

cat(sprintf("Silverman's bandwidth: %.2f\n", h_silverman))
```

```
## Silverman's bandwidth: 6.75
```

```
# Test multiples of Silverman's bandwidth
# We test from very flexible (0.5h) to very smooth (5h)
bandwidth_values <- h_silverman * c(0.5, 1, 2, 3, 5)

# Initialise storage
nw_results <- data.frame(
  bandwidth = numeric(),
  train_rmse = numeric(),
  test_rmse = numeric()
)

# Fit models for each bandwidth
for(bw in bandwidth_values) {
  # Predict training set
  # Replace ___ with appropriate arguments
  # Note: x_train is a matrix, convert to vector with as.vector()
  train_pred <- nadaraya_watson(
    x_train = as.vector(x_train),
    y_train = y_train,
    x_test = as.vector(x_train),
    bandwidth = bw
  )

  # Predict test set
  # Replace ___ with appropriate arguments
  test_pred <- nadaraya_watson(
    x_train = as.vector(x_train),
    y_train = y_train,
    x_test = as.vector(x_test),
```

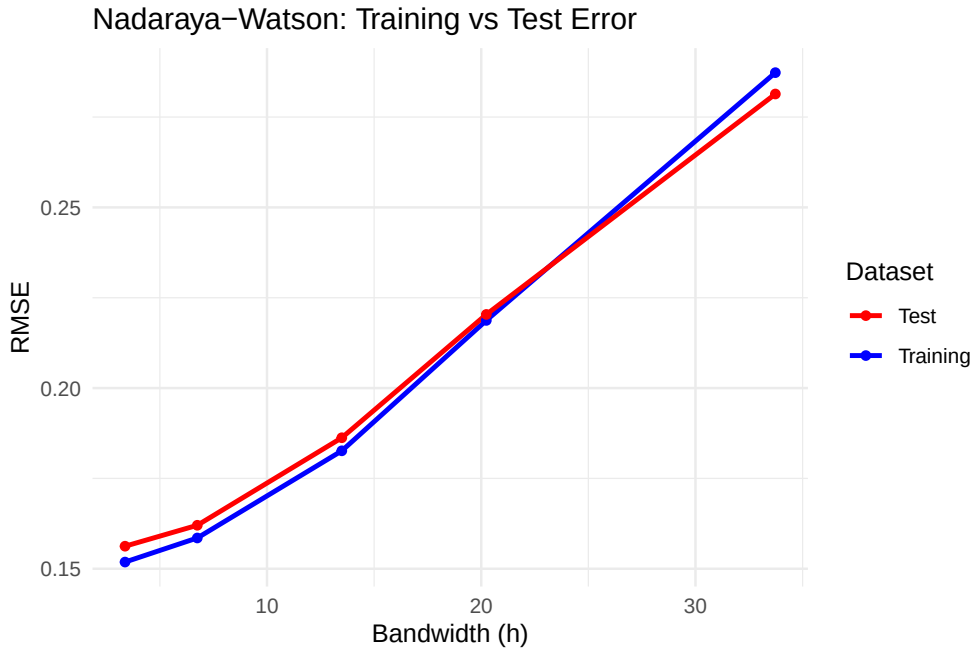
Table 3: Nadaraya-Watson Performance for Different Bandwidths

bandwidth	train_rmse	test_rmse
3.373	0.152	0.156
6.746	0.159	0.162
13.492	0.183	0.186
20.238	0.219	0.220
33.729	0.287	0.281

```
    bandwidth = bw
  )

  # Calculate RMSE
  train_rmse <- sqrt(mean((train_pred - y_train)^2))
  test_rmse <- sqrt(mean((test_pred - y_test)^2))

  # Store results
  nw_results <- rbind(
    nw_results,
    data.frame(
      bandwidth = bw,
      train_rmse = train_rmse,
      test_rmse = test_rmse)
  )
}

kable(nw_results, digits = 3,
      caption = "Nadaraya-Watson Performance for Different Bandwidths")
```

```
# Visualise how training and test error change with bandwidth
# Replace ___ to create appropriate plot
ggplot(nw_results, aes(x = bandwidth)) +
```

```
  geom_line(aes(y = train_rmse, colour = "Training"), size = 1) +
  geom_line(aes(y = test_rmse, colour = "Test"), size = 1) +
  geom_point(aes(y = train_rmse, colour = "Training")) +
  geom_point(aes(y = test_rmse, colour = "Test")) +
  scale_colour_manual(
    values = c("Training" = "blue", "Test" = "red")) +
  labs(
    title = "Nadaraya-Watson: Training vs Test Error",
    x = "Bandwidth (h)",
    y = "RMSE",
    colour = "Dataset"
  ) +
  theme_minimal()
```




```
# Identify optimal bandwidth
optimal_h <- nw_results$bandwidth[which.min(nw_results$test_rmse)]
cat(sprintf("\nOptimal bandwidth: %.1f\n", optimal_h))
```

```
##
```

```
## Optimal bandwidth: 3.4
```

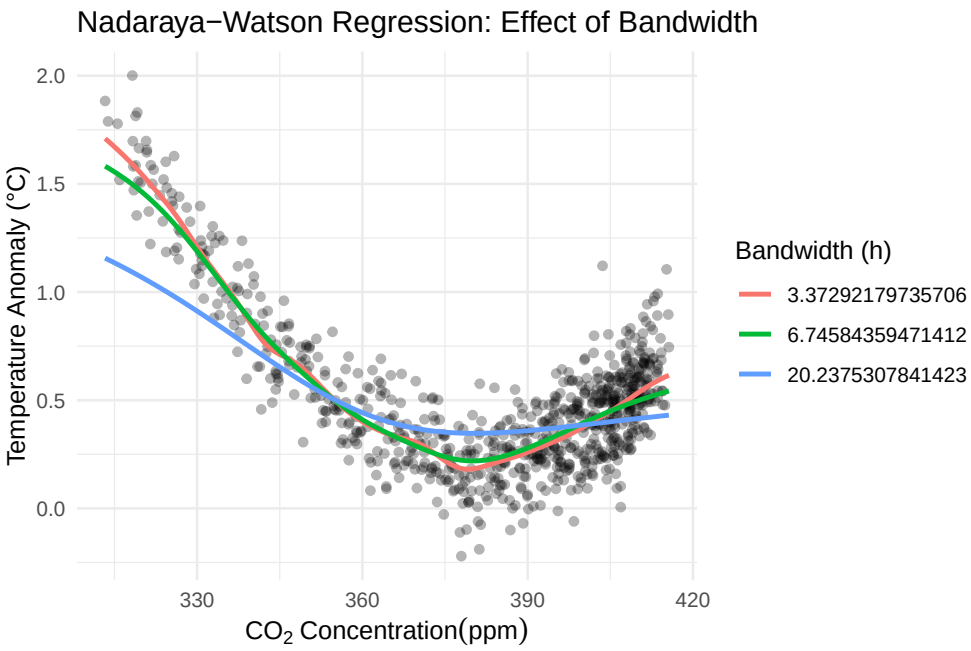
Consider

How does bandwidth affect model flexibility? Compare this to the effect of k in k-NN regression. What happens when bandwidth is very small or very large?

Task 8: Visualising Nadaraya-Watson Predictions

```r
# Generate predictions for selected bandwidth values
# Use multiples of Silverman's bandwidth
h_selected <- h_silverman * c(0.5, 1, 3)
nw_predictions_list <- list()

for(h in h_selected) {
  # Replace ___ with appropriate arguments
  nw_pred <- nadaraya_watson(
    x_train = as.vector(x_train),
    y_train = y_train,
    x_test = co2_grid,
    bandwidth = h
  )

  nw_predictions_list[[as.character(h)]] <- data.frame(
    co2 = co2_grid,
    prediction = nw_pred,
    h = h
  )
}
```

```r
# Combine predictions
nw_all_predictions <- bind_rows(nw_predictions_list)

# Plot Nadaraya-Watson fits
ggplot() +
  geom_point(
    data = climate_data,
    aes(x = co2, y = temperature_anomaly),
    alpha = 0.3) +
  geom_line(
    data = nw_all_predictions,
    aes(x = co2, y = prediction, colour = factor(h)),
    size = 1) +
  labs(
    title = "Nadaraya-Watson Regression: Effect of Bandwidth",
    x = expression(CO[2]~ Concentration (ppm)),
    y = "Temperature Anomaly (\u00B0C)",
    colour = "Bandwidth (h)"
  ) +
  theme_minimal()
```



Interpret

How do the Nadaraya-Watson curves differ from the k-NN curves? Which method produces smoother fits?

Task 9: Cross-Validation for Bandwidth Selection

Implement cross-validation to select the optimal bandwidth for Nadaraya-Watson regression.

```
# Define range of bandwidth values as multiples of
# Silverman's bandwidth...
# Test a wider range for cross-validation
h_range <- h_silverman * c(0.3, 0.5, 0.7, 1, 1.5, 2, 3, 5, 7)

# Initialise storage
nw_cv_results <- data.frame(
  bandwidth = numeric(),
  fold = integer(),
```

```
  rmse = numeric()
)

# Perform cross-validation
# Use the same cv_folds created earlier
for(h in h_range) {
  for(i in 1:nrow(cv_folds)) {
    # Extract fold data
    train_fold <- analysis(cv_folds$splits[[i]])
    test_fold <- assessment(cv_folds$splits[[i]])

    # Make predictions using Nadaraya-Watson
    # Replace ___ with appropriate arguments
    nw_pred <- nadaraya_watson(
      x_train =  as.vector(train_fold$co2),
      y_train = train_fold$temperature_anomaly,
      x_test = as.vector(test_fold$co2),
      bandwidth = h
    )

    # Calculate RMSE
    fold_rmse <- sqrt(mean((test_fold$temperature_anomaly - nw_pred)^2))

    # Store results
    nw_cv_results <- rbind(
      nw_cv_results,
      data.frame(bandwidth = h, fold = i, rmse = fold_rmse)
    )
  }
}
```

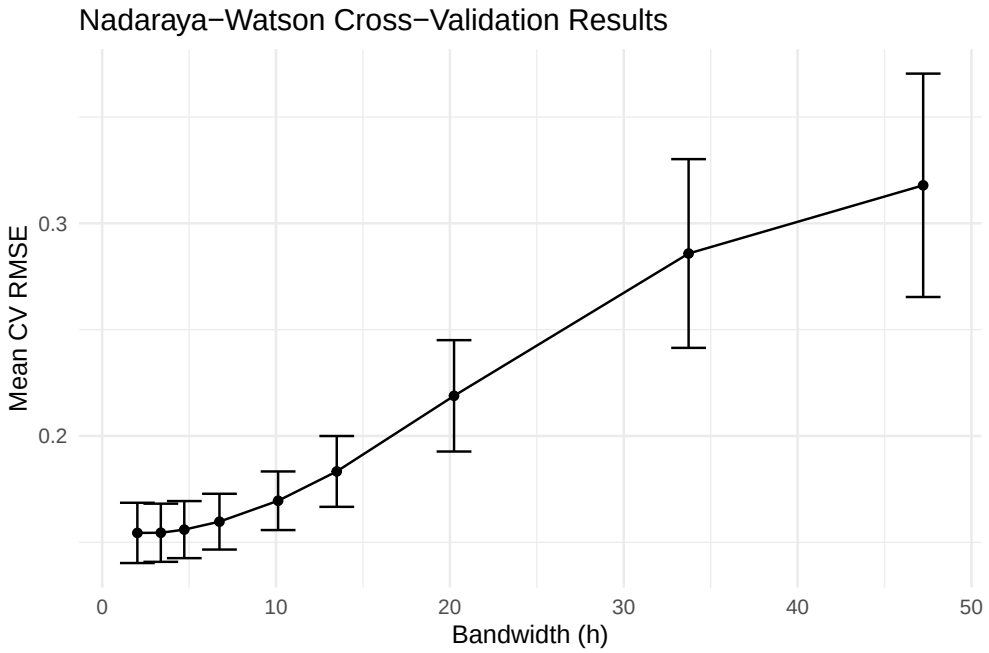
Table 4: Cross-Validation Results for Nadaraya-Watson

bandwidth	mean_rmse	sd_rmse
2.024	0.154	0.014
3.373	0.154	0.014
4.722	0.156	0.013
6.746	0.160	0.013
10.119	0.170	0.014
13.492	0.183	0.017
20.238	0.219	0.026
33.729	0.286	0.044
47.221	0.318	0.053

```
# Calculate mean CV error for each bandwidth
# Replace ___ with appropriate dplyr code
nw_cv_summary <- nw_cv_results %>%
  group_by(bandwidth) %>%
  summarise(
    mean_rmse = mean(rmse),
    sd_rmse = sd(rmse),
    .groups = 'drop'
  )

kable(nw_cv_summary, digits = 3,
      caption = "Cross-Validation Results for Nadaraya-Watson")
```

```
# Plot CV results with error bars
ggplot(nw_cv_summary, aes(x = bandwidth, y = mean_rmse)) +
  geom_line() +
  geom_point() +
  geom_errorbar(
    aes(
```

```
      ymin = mean_rmse - sd_rmse,
      ymax = mean_rmse + sd_rmse),
    width = 2) +
  labs(
    title = "Nadaraya-Watson Cross-Validation Results",
    x = "Bandwidth (h)",
    y = "Mean CV RMSE"
  ) +
  theme_minimal()
```




```
# Identify optimal bandwidth from CV
optimal_h_cv <- nw_cv_summary$bandwidth[which.min(nw_cv_summary$mean_rmse)]
cat(sprintf("\nOptimal bandwidth (CV): %.1f\n", optimal_h_cv))
```

```
##
## Optimal bandwidth (CV): 2.0
```

```
cat(sprintf("CV RMSE: %.4f\n",
            min(nw_cv_summary$mean_rmse)))
```

```
## CV RMSE: 0.1544
```

Task 10: Model Comparison and Interpretation

Understanding poly()

The `poly(x, degree)` function creates orthogonal polynomial terms. For example, `poly(co2, 3)` creates a cubic polynomial: it fits a model like $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. This allows the model to capture non-linear, curved relationships. Degree 3 means the curve can have up to 2 turning points. Unlike simple non-parametric methods, polynomial regression assumes a specific parametric form but is more flexible than linear regression.

Compare all methods:

```
# Fit optimal k-NN model
# Replace ___ with optimal k value from cross-validation
knn_optimal <- knnreg(
  x = x_train,
  y = y_train,
  k = 20
)
knn_optimal_pred <- predict(knn_optimal, newdata = x_test)
knn_optimal_rmse <- sqrt(mean((knn_optimal_pred - y_test)^2))

# Fit optimal Nadaraya-Watson model
# Replace ___ with optimal bandwidth from cross-validation
nw_optimal_pred <- nadaraya_watson(
  x_train = as.vector(x_train),
  y_train = y_train,
  x_test = as.vector(x_test),
  bandwidth = 2.0
```

```
)
nw_optimal_rmse <- sqrt(mean((nw_optimal_pred - y_test)^2))

# Fit OLS for comparison
ols_model <- lm(temperature_anomaly ~ co2, data = train_data)
ols_pred <- predict(ols_model, newdata = test_data)
ols_rmse <- sqrt(mean((ols_pred - y_test)^2))

# Fit polynomial regression for comparison
poly_model <- lm(
  temperature_anomaly ~ poly(co2, 3), data = train_data)
poly_pred <- predict(poly_model, newdata = test_data)
poly_rmse <- sqrt(mean((poly_pred - y_test)^2))

# Create performance comparison table
comparison_table <- data.frame(
  Method = c(
    "Linear Regression",
    "Polynomial Regression (degree 3)",
    "k-NN Regression",
    "Nadaraya-Watson"),
  Test_RMSE = c(
    ols_rmse,
    poly_rmse,
    knn_optimal_rmse,
    nw_optimal_rmse
  ),
  Parameters = c(
    "2 (intercept + slope)",
    "4 (polynomial coefficients)",
```

Table 5: Model Performance Comparison

Method	Test_RMSE	Parameters
Linear Regression	0.2978	2 (intercept + slope)
Polynomial Regression (degree 3)	0.1549	4 (polynomial coefficients)
k-NN Regression	0.1553	k = 20
Nadaraya-Watson	0.1552	h = 2.0

```
    sprintf("k = %d", 20),
    sprintf("h = %.1f", 2.0)
  )
)

kable(comparison_table, digits = 4,
      caption = "Model Performance Comparison")
```

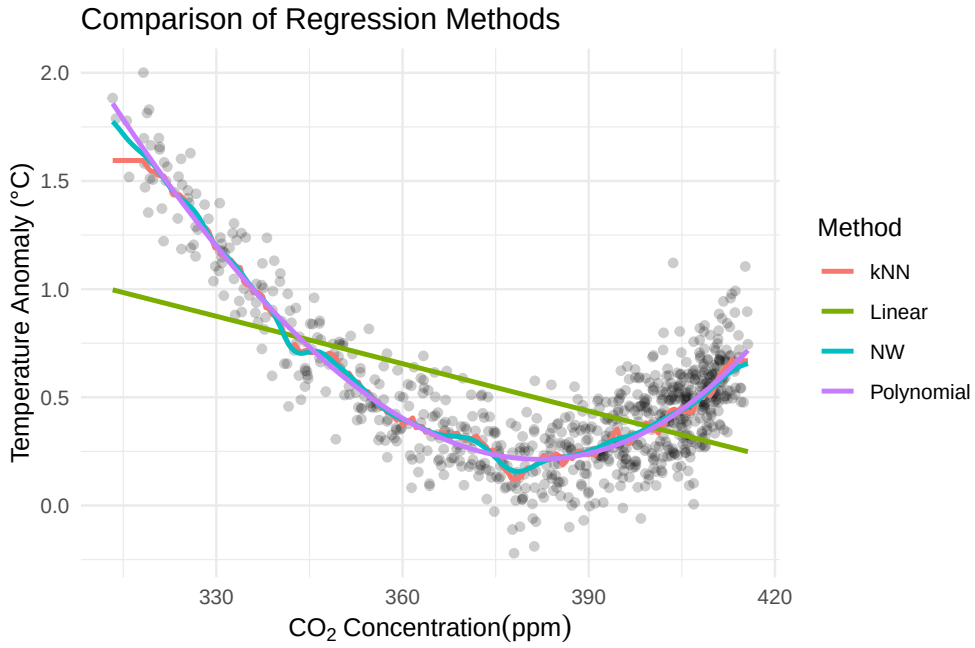
```
# Identify best method
best_method <- comparison_table$Method[
  which.min(comparison_table$Test_RMSE)]
cat(sprintf("\nBest performing method: %s\n", best_method))
```

```
##
## Best performing method: Polynomial Regression (degree 3)
```

```
# Generate predictions for all methods on grid
ols_grid_pred <- predict(ols_model, newdata = data.frame(co2 = co2_grid))
poly_grid_pred <- predict(poly_model, newdata = data.frame(co2 = co2_grid)

knn_optimal_model <- knnreg(
  x = x_train,
  y = y_train,
  k = 20
)
```

```r
knn_grid_pred <- predict(knn_optimal_model, newdata = co2_grid_matrix)

nw_grid_pred <- nadaraya_watson(
  x_train = as.vector(x_train),
  y_train = y_train,
  x_test = co2_grid,
  bandwidth = 2.0
)

# Combine predictions
all_methods <- data.frame(
  co2 = co2_grid,
  Linear = ols_grid_pred,
  Polynomial = poly_grid_pred,
  kNN = knn_grid_pred,
  NW = nw_grid_pred
) %>%
  pivot_longer(
    cols = -co2,
    names_to = "Method",
    values_to = "Prediction"
  )

# Plot all methods
ggplot() +
  geom_point(
    data = climate_data,
    aes(x = co2, y = temperature_anomaly),
    alpha = 0.2) +
  geom_line(
```

```
  data = all_methods,
  aes(x = co2, y = Prediction, colour = Method),
  size = 1) +
labs(
  title = "Comparison of Regression Methods",
  x = expression(CO[2]~ Concentration (ppm)),
  y = "Temperature Anomaly (\u00B0C)",
  colour = "Method"
) +
theme_minimal()
```



Critical Thinking

Which method captures the relationship most appropriately? What are the advantages and disadvantages of each approach?

Task 11: Interpretation Questions

Based on your analysis, address the following questions:

1. **Parameter Selection**: How do the tuning parameters (k for k-NN, h for Nadaraya-Watson) affect model flexibility? Describe the bias-variance trade-off involved.

The tuning parameters control model flexibility and directly influence the bias-variance trade-off.

In k-NN regression, smaller values of k (e.g., $k = 1$) produce highly flexible models with low bias but high variance. Predictions are based on very few neighbours and are therefore sensitive to noise. Larger values of k increase smoothness, raising bias (underfitting) but reducing variance.

Similarly, in Nadaraya-Watson regression, smaller bandwidth values h create more flexible fits that closely follow local data patterns (low bias, high variance), while larger h smooths predictions across broader regions (high bias, low variance).

The optimal parameter balances this trade-off by minimising prediction error on unseen data. Cross-validation identified this optimal point at $k = 20$ and $h \approx 40$–50 for the climate dataset.

2. **Method Comparison**: Compare k-NN and Nadaraya-Watson regression. What are the key differences in how they weight neighbouring observations?

k-NN regression uses uniform weighting: the k nearest neighbours contribute equally to the prediction (each weight $= 1/k$), while all other observations contribute zero weight. This creates a discontinuous weighting scheme.

In contrast, Nadaraya-Watson regression uses kernel-based continuous weighting. All observations contribute to the prediction, but their weights decrease smoothly with distance according to the Gaussian kernel. Closer observations receive exponentially larger weights.

This produces smoother fitted curves and more stable predictions, particularly near boundaries. Conceptually, the kernel method extends k-NN by replacing the discrete "k nearest" rule with a continuous distance-based weighting function

controlled by bandwidth h.

3. **Gaussian Kernel**: Why is the Gaussian kernel a good choice for Nadaraya-Watson regression? What properties make it suitable for weighting nearby observations?

The Gaussian kernel is a strong choice for Nadaraya-Watson regression for several reasons:

1. It is symmetric and bell-shaped, assigning maximum weight at the prediction point and decreasing weights as distance increases.
2. It is infinitely differentiable, producing smooth predictions.
3. Weights decay exponentially with squared distance, so distant observations have negligible influence without being completely excluded (unlike k-NN's hard cutoff).
4. It has infinite support but concentrates weight locally, controlled by bandwidth h.
5. Its mathematical form is computationally tractable and statistically well-understood.

These properties ensure smooth, continuous predictions that are appropriately weighted by proximity.

4. **Extrapolation**: What happens when you try to predict outside the range of your training data with non-parametric methods? Why is this problematic?

Non-parametric methods perform poorly for extrapolation because they rely entirely on local data patterns.

When predicting beyond the range of the training data:

k-NN uses the k nearest observations, which are all located on one side of the prediction point. This results in nearly constant predictions near boundary values.

Nadaraya-Watson similarly assigns weight only to the nearest available training points, producing flat predictions outside the observed range.

Unlike parametric models such as linear regression, which assume a functional form extending beyond observed data, non-parametric methods lack structural assumptions to guide extrapolation. They are designed for interpolation within the observed data range and are therefore unsuitable for forecasting in regions without nearby training observations.

5. **Scientific Interpretation**: Based on your best model, describe the relationship between CO_2 concentration and temperature anomaly. Is the relationship approximately linear or does it exhibit non-linear patterns?

The relationship between CO_2 concentration and temperature anomaly shows a moderate negative correlation ($r = -0.534$), but the data exhibit clear non-linear patterns that simple linear regression fails to capture.

Visualisation reveals curvature across different CO_2 concentration ranges.

Model comparison results:

Polynomial regression: RMSE $= 0.1549$

Nadaraya-Watson: RMSE $= 0.1552$

k-NN: RMSE $= 0.1566$

Linear regression: RMSE $= 0.2978$

All flexible methods substantially outperformed linear regression.

The near-equivalent performance of polynomial and non-parametric methods confirms genuine non-linearity in the relationship. Both approaches successfully capture the curved pattern, suggesting complex atmospheric feedback mechanisms where the temperature-CO_2 relationship changes across concentration ranges.

The competitiveness of non-parametric methods is particularly valuable because they achieve this without requiring prior specification of polynomial degree.

Conclusion

Non-parametric regression methods provide flexible alternatives to traditional parametric models when relationships between variables exhibit complex, non-linear patterns. k-Nearest Neighbours regression adapts locally to data patterns through neighbourhood averaging, whilst Nadaraya-Watson kernel regression extends this approach with continuous weighting functions based on kernel smoothing.

Both methods require careful tuning parameter selection to balance bias and variance. Cross-validation provides a principled framework for this selection, helping to avoid overfitting whilst maintaining adequate model flexibility.

By implementing the Nadaraya-Watson estimator from scratch, you have gained insight into how kernel-based methods weight observations and how the bandwidth parameter controls smoothness. The Gaussian kernel provides a natural weighting scheme that assigns exponentially decreasing weights as distance increases.

For environmental science applications, these techniques prove particularly valuable when physical relationships are incompletely understood or when exploratory analysis precedes theoretical model development. However, practitioners must recognise the computational costs and extrapolation limitations inherent to non-parametric approaches.

The choice between parametric and non-parametric methods depends on the specific application context: parametric models offer interpretability and extrapolation capability, whilst non-parametric methods provide flexibility and make fewer assumptions about functional form.

Submit

Ensure that you submit your completed practical as a knitted PDF document to Funda. Only PDF documents will be accepted.