CW2

2025-04-08

Introduction

This project focuses on developing predictive models to estimate near-surface air temperature (`t2m`) using a large-scale meteorological dataset. The main goal is to explore how different machine learning approaches perform when applied to spatio-temporal weather data, and to evaluate their effectiveness in capturing temperature variation across time and location.

To approach this task, we apply a consistent preprocessing pipeline and train a range of supervised learning models. Specifically, we fit, tune and assess the performance of the following regression models:

- Decision Tree Regressor
- Bagging ensemble of decision trees
- Random Forest
- Neural Network

Each model is evaluated using metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), allowing for a comparative analysis of their predictive accuracy. This document outlines the preprocessing steps, modeling approaches, and evaluation results that contribute to selecting the most suitable method for the temperature prediction task.

Dataset description?

The dataset used in this project comes from the ERA5 reanalysis dataset, provided by the European Centre for Medium-Range Weather Forecasts (ECMWF). It includes hourly weather observations for the years 2018 and 2019, covering a grid of latitude and longitude points approximately spanning the United Kingdom.

Each record in the dataset corresponds to a unique combination of time and location, and contains various meteorological variables such as: • t2m – Temperature at 2 meters above ground (in Kelvin), the target variable for prediction • tp – Total precipitation • sp – Surface pressure • u10, v10 – Wind components at 10 meters (east-west and north-south) • u100, v100 – Wind components at 100 meters • tcc – Total cloud cover • ptype – Precipitation type (e.g., rain, snow)

The dataset exhibits strong temporal patterns (e.g., daily and seasonal cycles) and spatial structure (via latitude and longitude), making it well-suited for predictive modeling that considers both time and location. It contains over 13 million rows in both the training and test sets, making computational efficiency an important consideration throughout the analysis.

Structure and summary

```
# Structure and summary
str(train)
```

```
## 'data.frame':    13288920 obs. of  13 variables:
##  $ id        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ valid_time: chr  "2018-01-01 00:00:00" "2018-01-01 00:00:00" "2018-01-01 00:00:00" "2018-01-01 00
```

```
## $ latitude  : num  59 59 59 59 59 59 59 59 59 59 ...
## $ longitude : num  -8 -7.75 -7.5 -7.25 -7 -6.75 -6.5 -6.25 -6 -5.75 ...
## $ tp        : num  3.81e-06 1.67e-05 1.81e-05 1.91e-05 1.53e-05 ...
## $ u10       : num  1.41 2.15 2.69 2.85 3.3 ...
## $ v10       : num  -1.37 -1.56 -1.59 -1.71 -1.87 ...
## $ sp        : num  98135 98087 98079 98077 98085 ...
## $ u100      : num  1.52 2.24 2.78 2.97 3.46 ...
## $ v100      : num  -1.47 -1.63 -1.69 -1.85 -2 ...
## $ tcc       : num  0.969 0.979 0.976 0.979 0.969 ...
## $ ptype     : num  1 1 1 1 1 1 1 1 1 1 ...
## $ t2m       : num  280 280 280 280 280 ...
```

```r
summary(train)
```

```
##        id            valid_time           latitude       longitude
##  Min.   :       1   Length:13288920    Min.   :50.00   Min.   :-8.0
##  1st Qu.: 3322231   Class :character   1st Qu.:52.25   1st Qu.:-5.5
##  Median : 6644460   Mode  :character   Median :54.50   Median :-3.0
##  Mean   : 6644460                      Mean   :54.50   Mean   :-3.0
##  3rd Qu.: 9966690                      3rd Qu.:56.75   3rd Qu.:-0.5
##  Max.   :13288920                      Max.   :59.00   Max.   : 2.0
##        tp                 u10               v10                sp
##  Min.   :0.000e+00   Min.   :-19.563   Min.   :-19.074   Min.   : 90686
##  1st Qu.:0.000e+00   1st Qu.: -2.052   1st Qu.: -1.637   1st Qu.: 99632
##  Median :3.815e-06   Median :  1.158   Median :  1.341   Median :100809
##  Mean   :1.154e-04   Mean   :  1.055   Mean   :  1.496   Mean   :100530
##  3rd Qu.:6.437e-05   3rd Qu.:  4.234   3rd Qu.:  4.463   3rd Qu.:101707
##  Max.   :1.099e-02   Max.   : 24.990   Max.   : 23.159   Max.   :104372
##       u100              v100               tcc              ptype
##  Min.   :-24.096   Min.   :-24.368   Min.   :0.0000   Min.   :0.000
##  1st Qu.: -2.824   1st Qu.: -2.273   1st Qu.:0.4150   1st Qu.:0.000
##  Median :  1.856   Median :  1.808   Median :0.8707   Median :1.000
##  Mean   :  1.569   Mean   :  2.100   Mean   :0.6968   Mean   :0.803
##  3rd Qu.:  6.025   3rd Qu.:  6.249   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   : 31.781   Max.   : 29.405   Max.   :1.0000   Max.   :8.000
##       t2m
##  Min.   :258.9
##  1st Qu.:279.8
##  Median :283.2
##  Mean   :283.3
##  3rd Qu.:286.7
##  Max.   :308.0
```

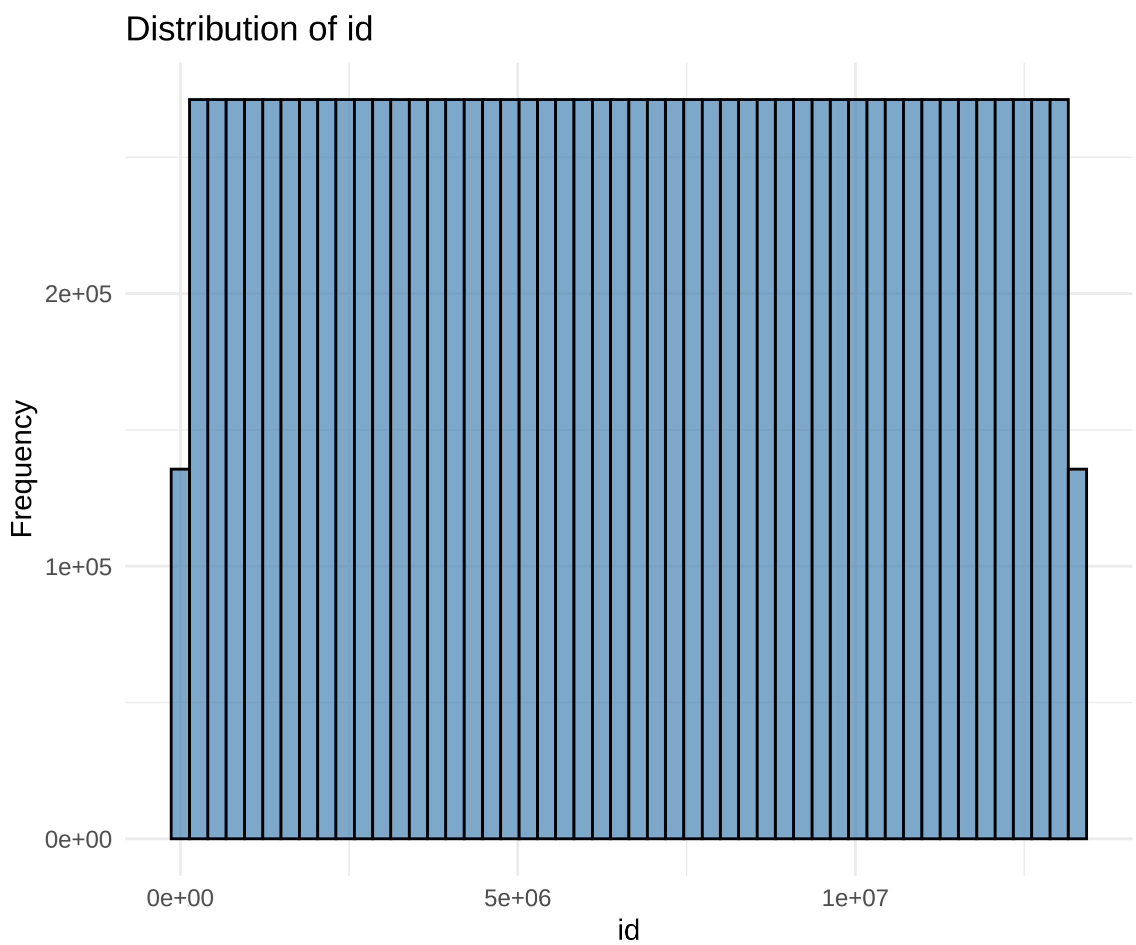
NAs?

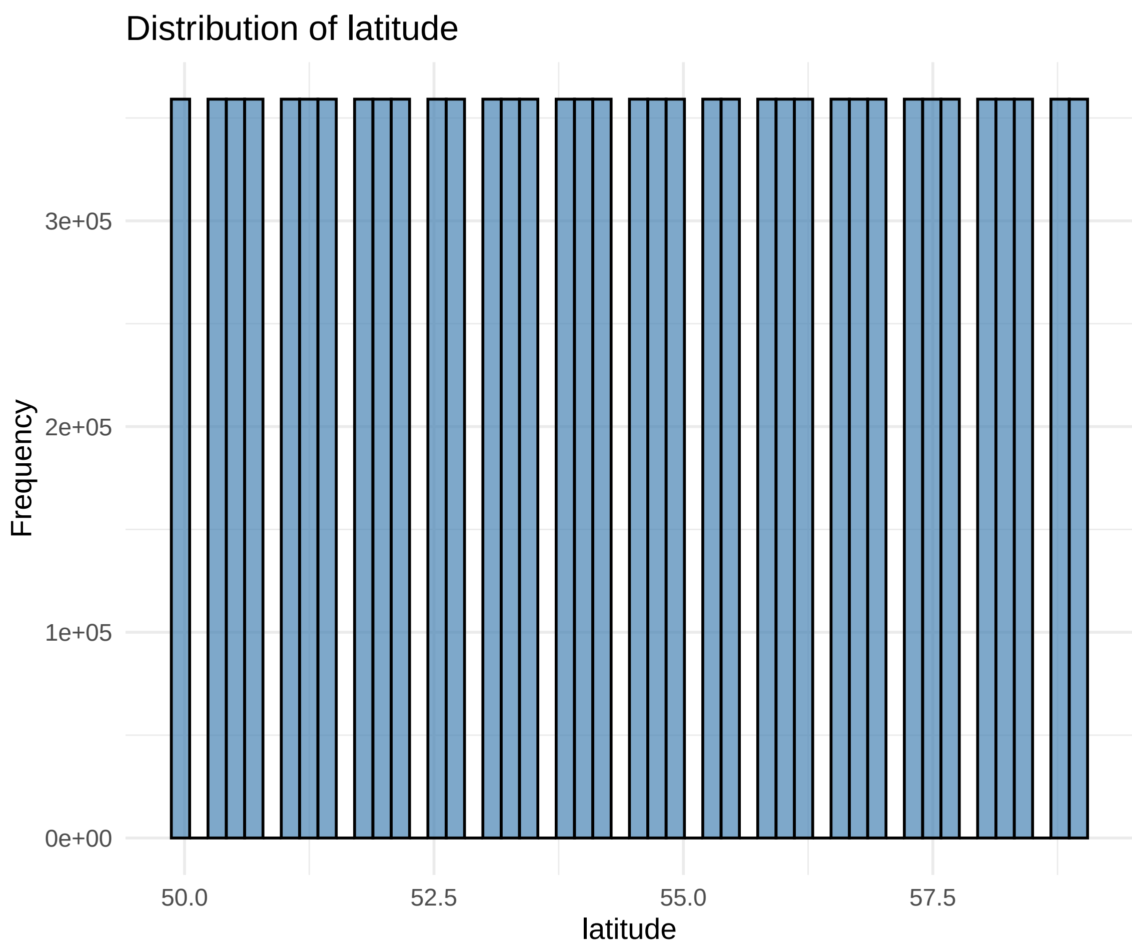
```r
colSums(is.na(train))
```

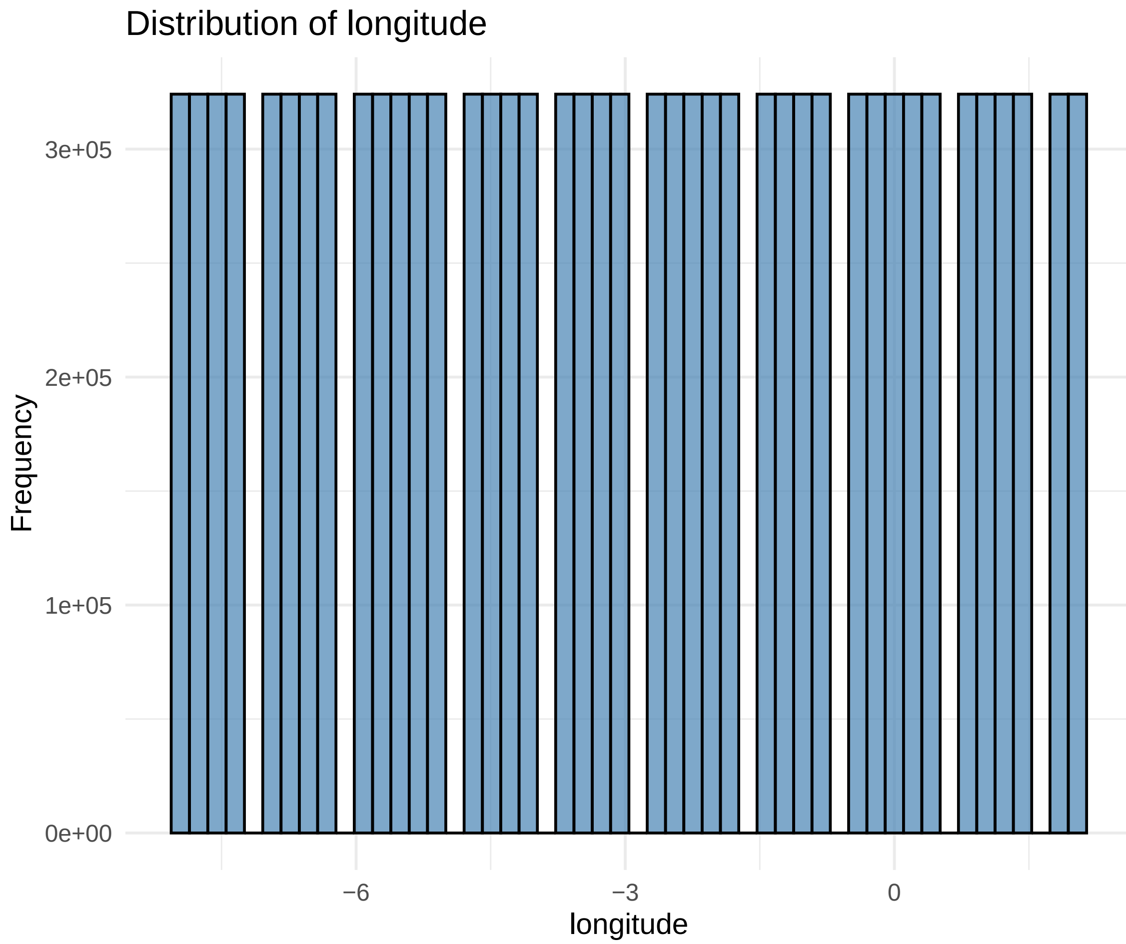
```
##         id valid_time   latitude  longitude         tp        u10        v10
##          0          0          0          0          0          0          0
##         sp       u100       v100        tcc      ptype        t2m
##          0          0          0          0          0          0
```

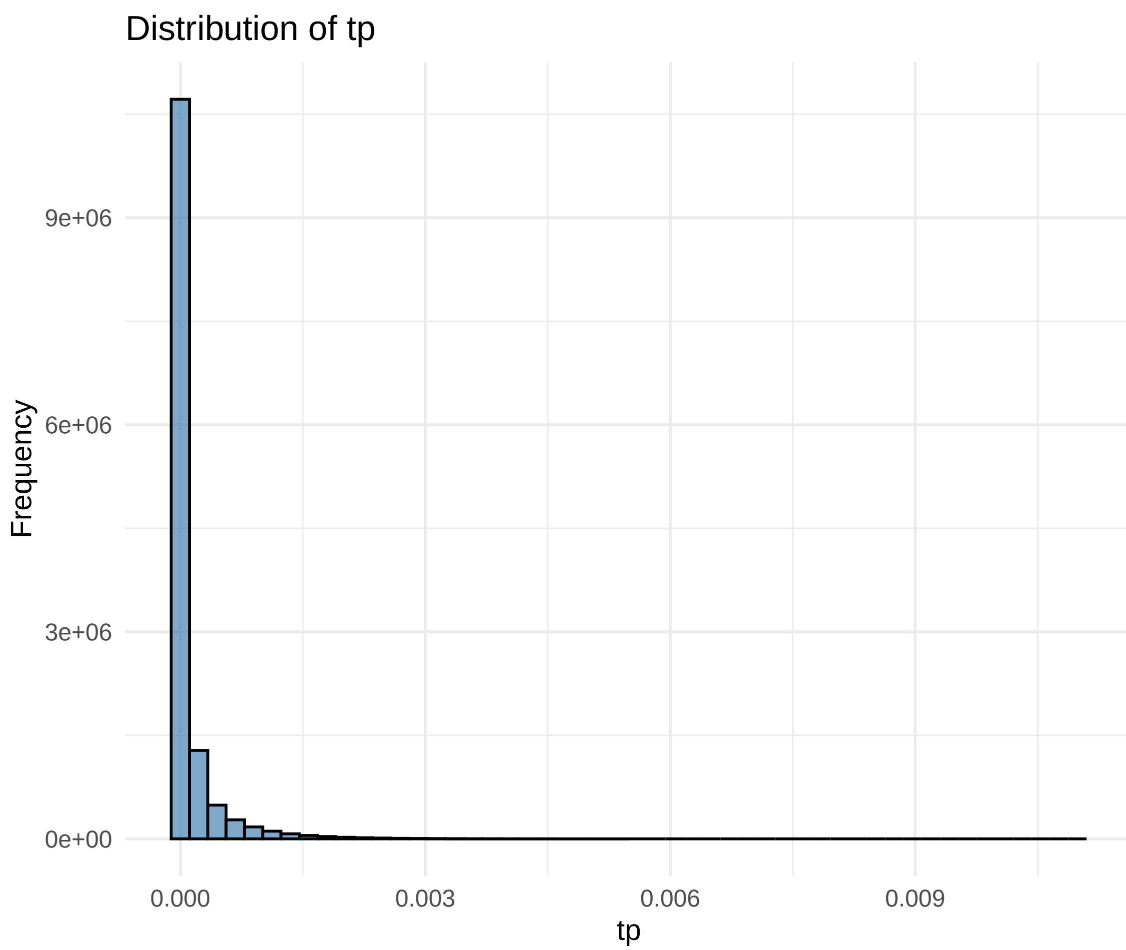
Distributions

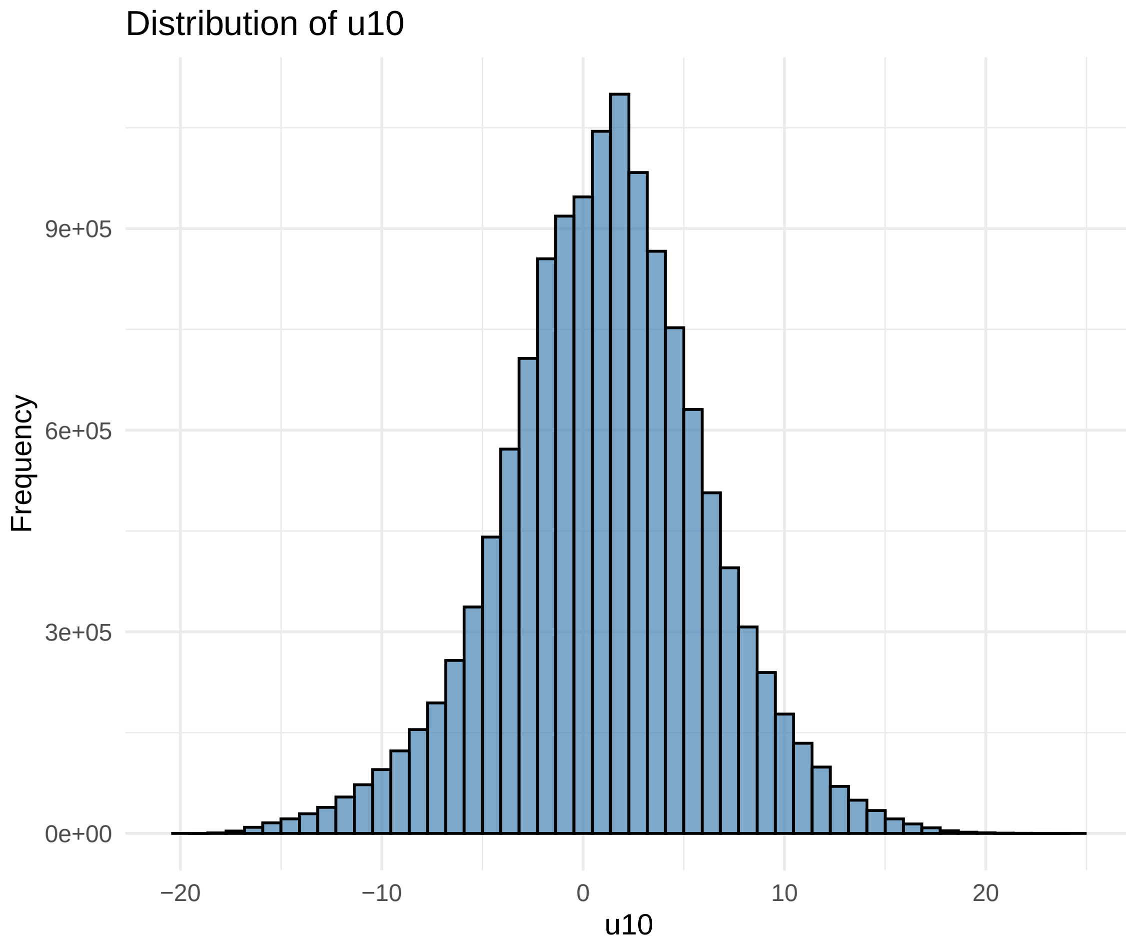
```r
library(ggplot2)
library(tidyr)
library(dplyr)

# Select numeric variables
numeric_vars <- train %>%
  select(where(is.numeric)) %>%
  names()

# Plot distributions
for (var in numeric_vars) {
  print(
    ggplot(train, aes(x = .data[[var]])) +
      geom_histogram(bins = 50, fill = "steelblue", color = "black", alpha = 0.7) +
      labs(title = paste("Distribution of", var), x = var, y = "Frequency") +
      theme_minimal()
  )
}
```

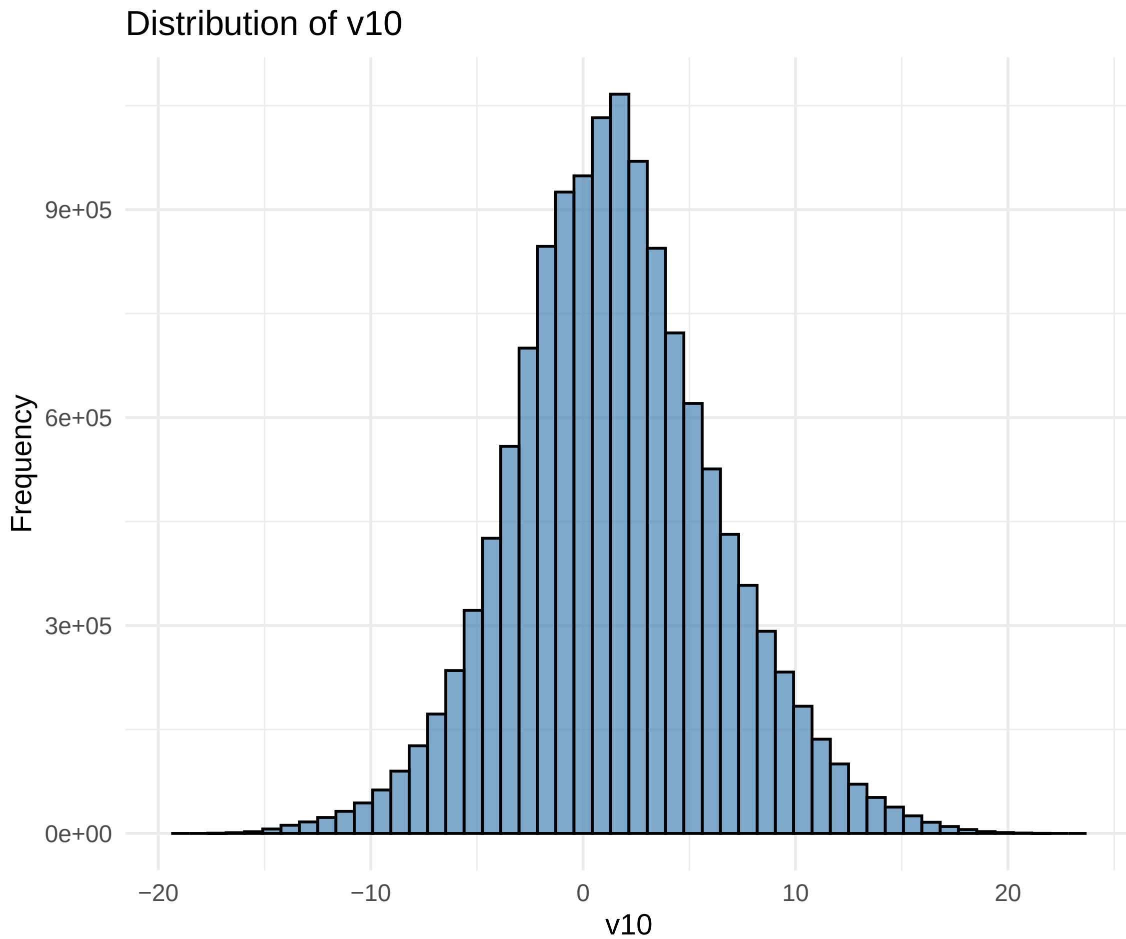


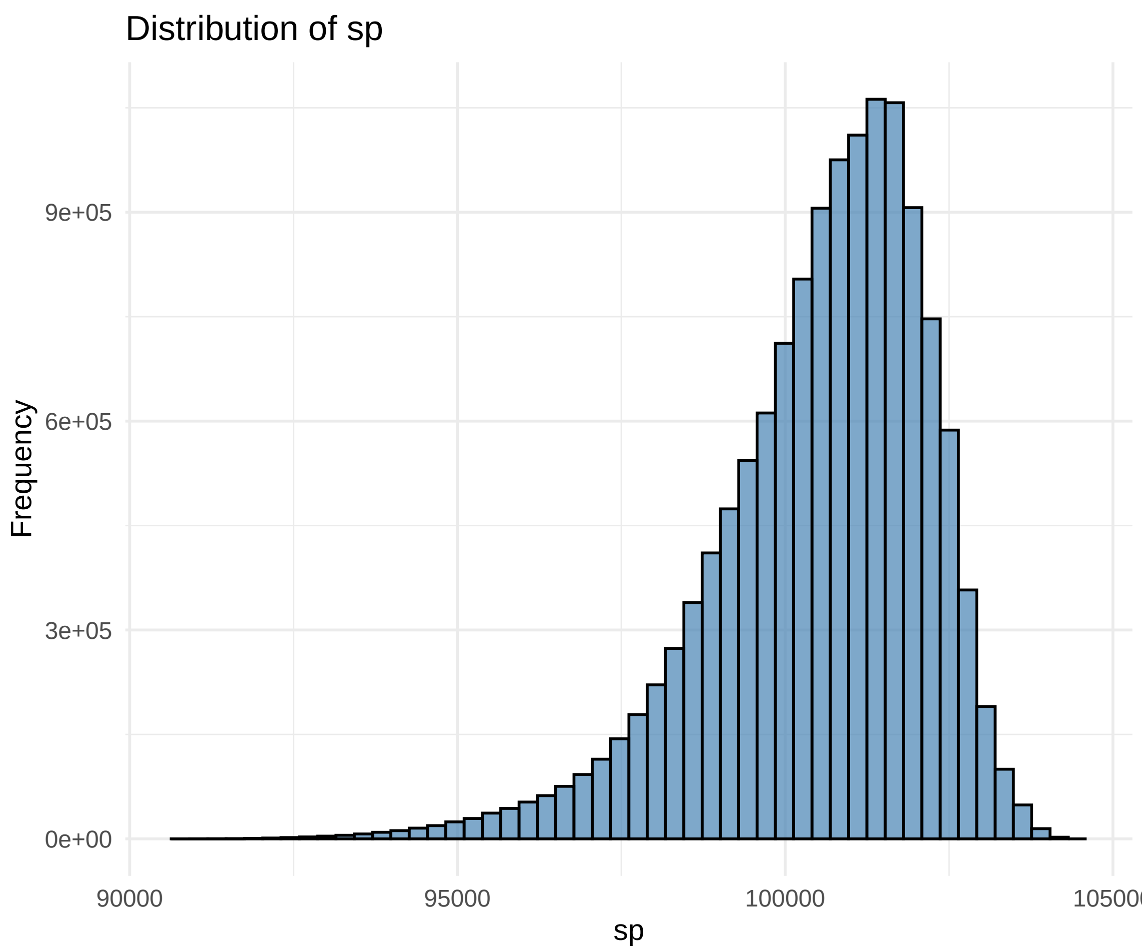

Distribution of latitude
Frequency
3e+05
2e+05
1e+05
0e+00
50.0
52.5
55.0
57.5
latitude

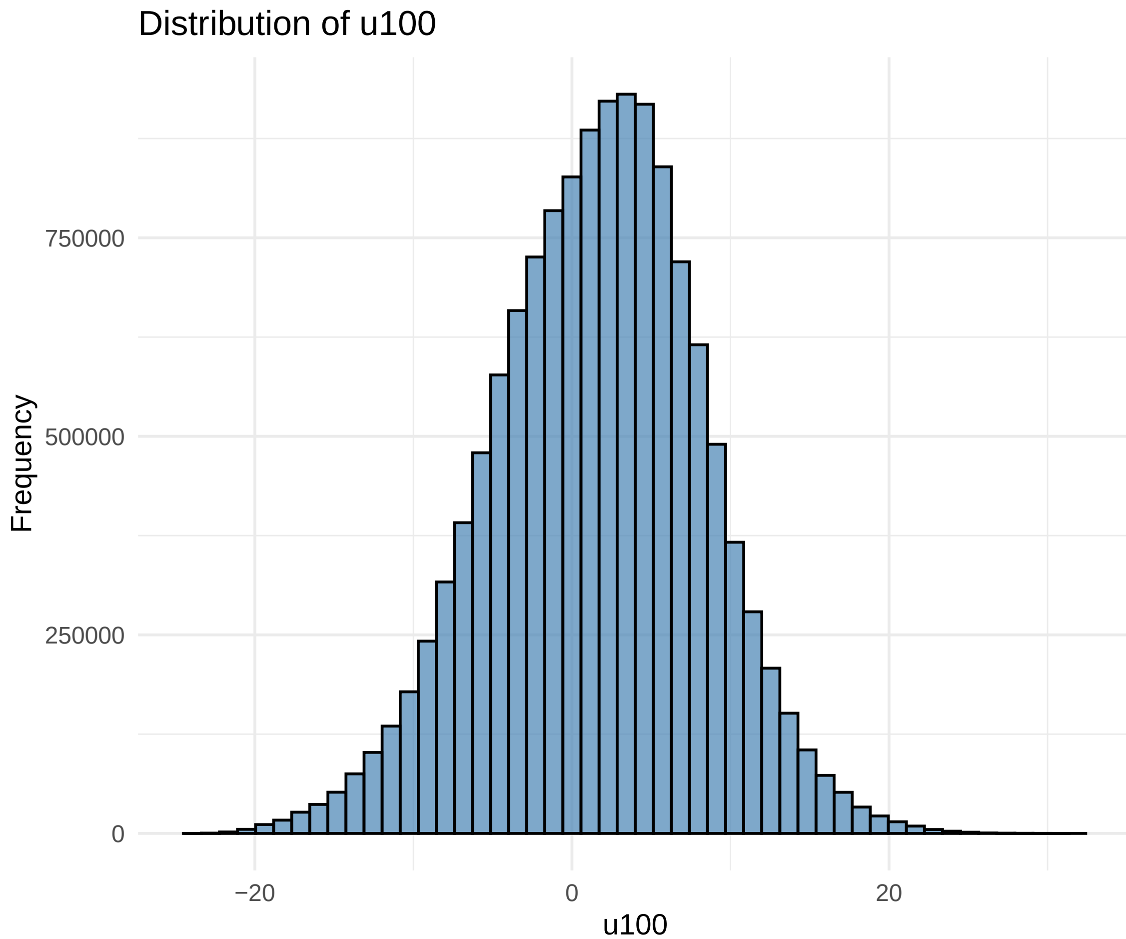

Distribution of longitude
3e+05
2e+05
1e+05
0e+00
Frequency
−6
−3
0
longitude

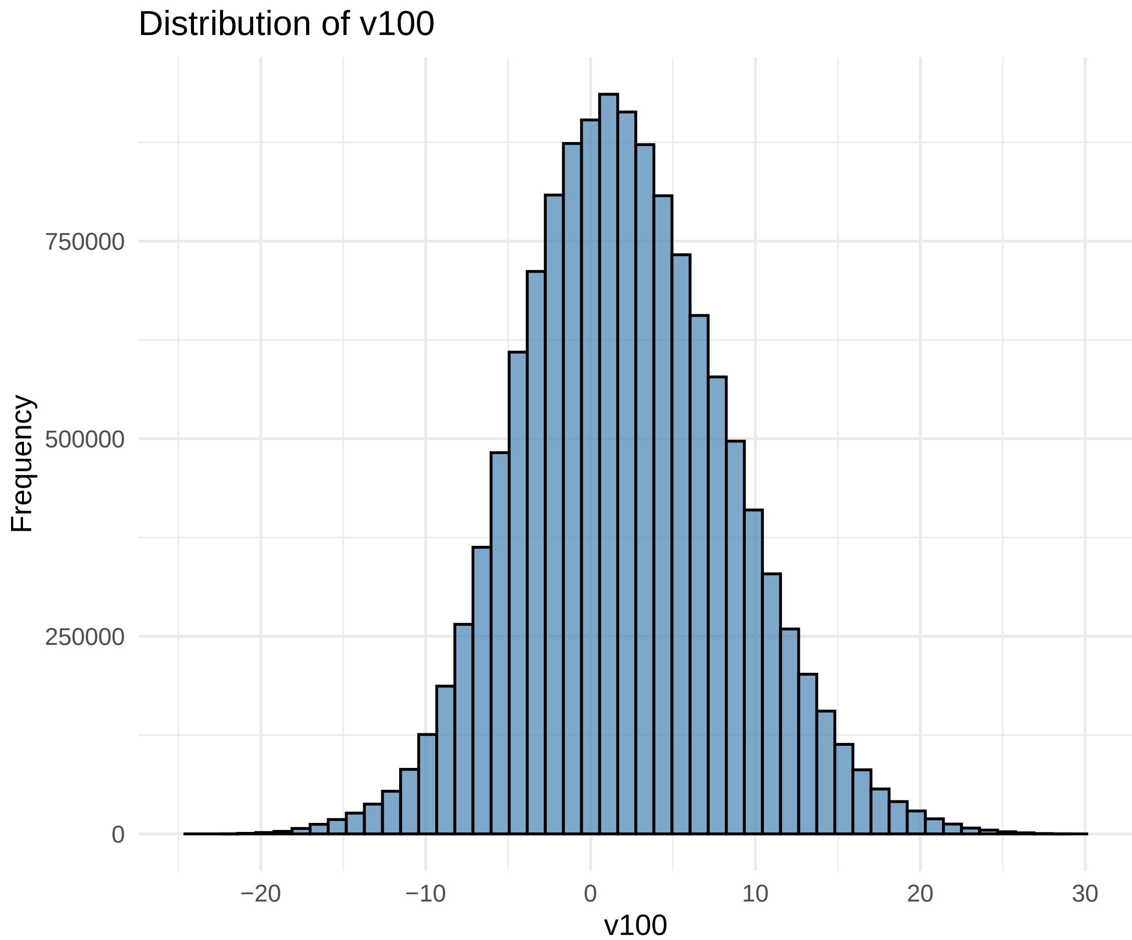

Distribution of tp
Frequency
9e+06
6e+06
3e+06
0e+00
0.000
0.003
0.006
0.009
tp

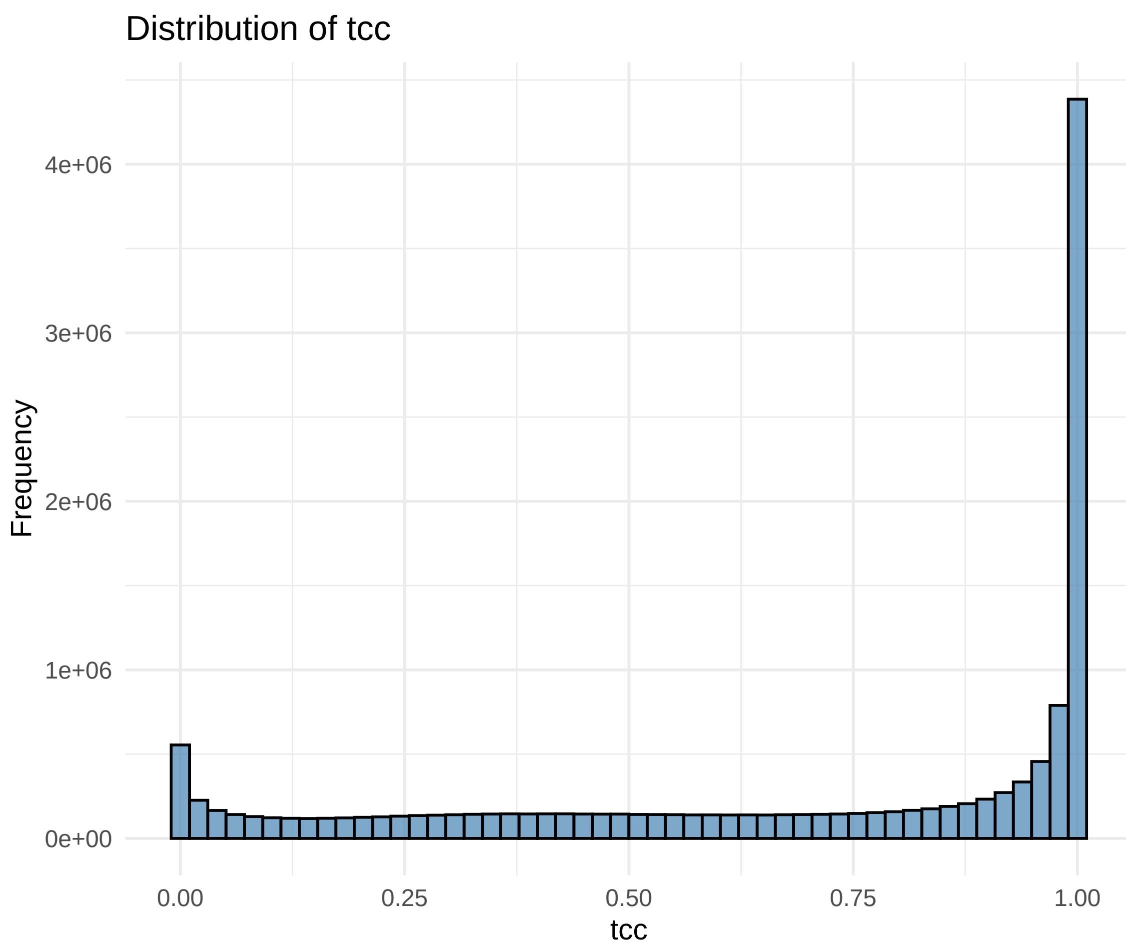

Distribution of u10
Frequency
9e+05
6e+05
3e+05
0e+00
−20
−10
0
10
20
u10


Distribution of v10
9e+05
6e+05
3e+05
0e+00
Frequency
−20
−10
0
10
20
v10

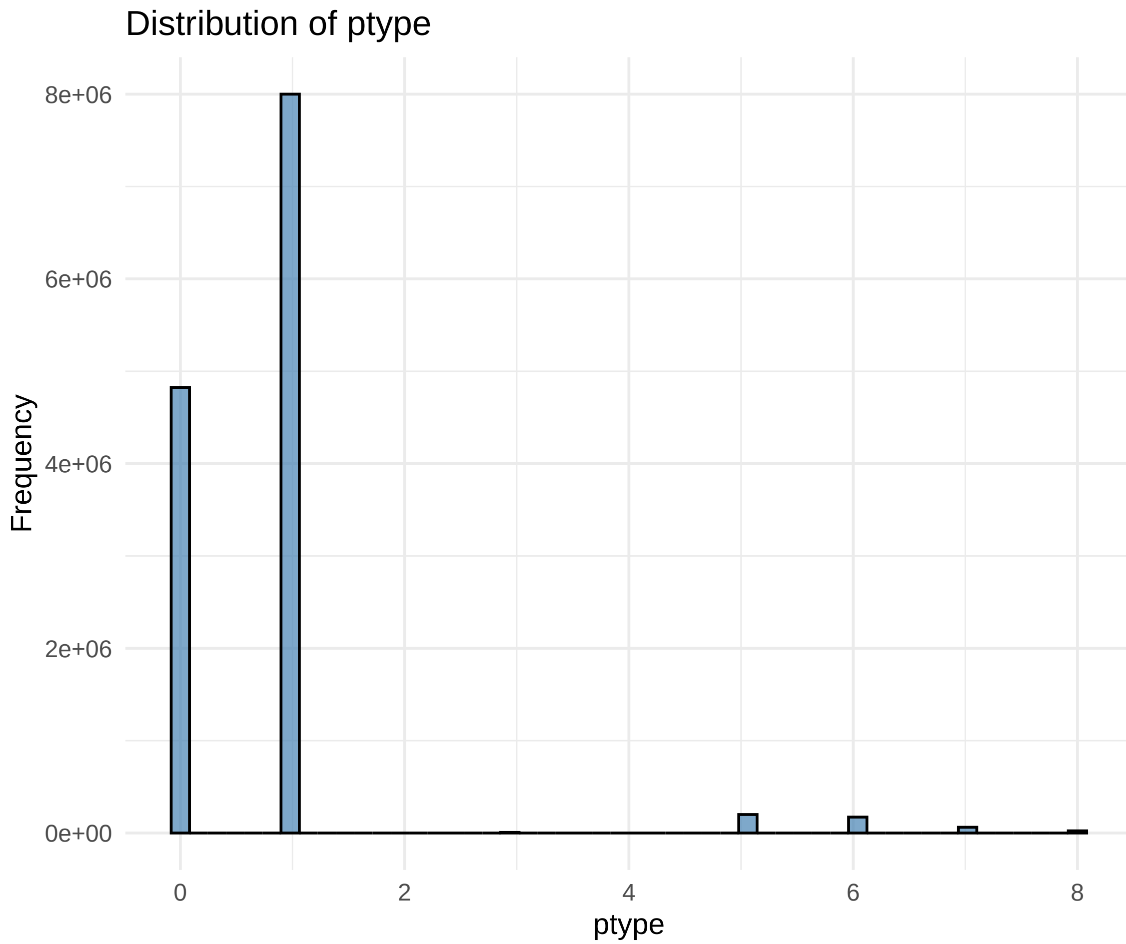

Distribution of sp
Frequency
9e+05
6e+05
3e+05
0e+00
90000
95000
100000
105000
sp

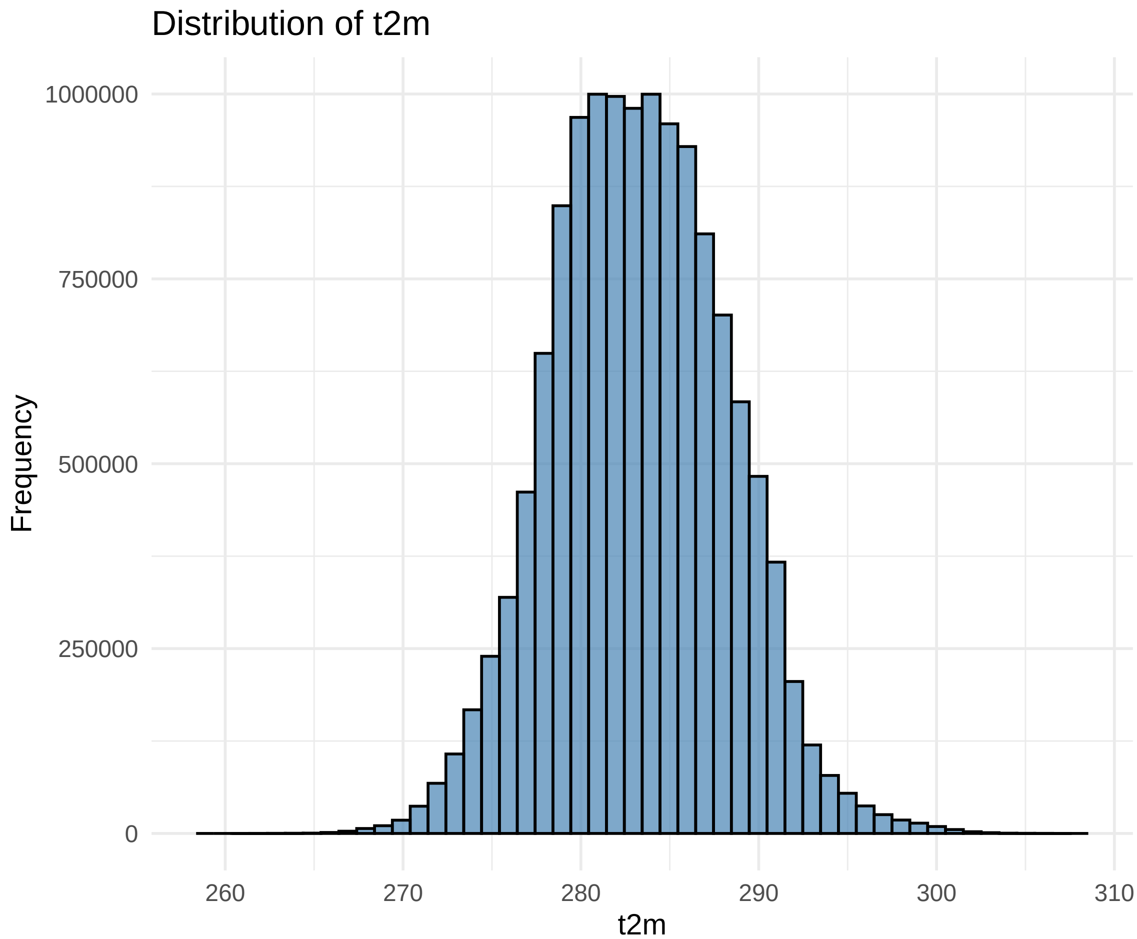

Distribution of u100
Frequency
750000
500000
250000
0
−20
0
20
u100


Distribution of v100
Frequency
750000
500000
250000
0
−20
−10
0
10
20
30
v100



Distribution of tcc
Frequency
4e+06
3e+06
2e+06
1e+06
0e+00
0.00
0.25
0.50
0.75
1.00
tcc


Distribution of ptype
8e+06
6e+06
4e+06
2e+06
0e+00
Frequency
0
2
4
6
8
ptype



ID

The id column is removed because it is simply a unique identifier and carries no predictive value. Keeping it could introduce noise or misleading patterns into the model

```r
# Drop the 'id' column
train$id <- NULL
```

Date

We extract hour, month, and dayofweek from valid_time to capture temporal patterns in temperature. These features help the model learn daily and seasonal cycles in the data

```r
library(lubridate)
library(dplyr)

# Convert to datetime format
train$valid_time <- ymd_hms(train$valid_time)

# Extract time-based features
train$hour <- hour(train$valid_time)
train$month <- month(train$valid_time)
train$dayofweek <- wday(train$valid_time, label = FALSE) - 1  # Make it 0 = Monday
```

```r
# Fix Sunday wraparound (wday returns 1 = Sunday by default)
train$dayofweek[train$dayofweek == -1] <- 6

# Display some examples
train %>%
  select(valid_time, hour, month, dayofweek) %>%
  sample_n(10)
```

```
##             valid_time hour month dayofweek
## 1  2018-10-05 01:00:00    1    10         5
## 2  2018-06-09 10:00:00   10     6         6
## 3  2018-01-23 20:00:00   20     1         2
## 4  2018-10-10 15:00:00   15    10         3
## 5  2018-04-03 23:00:00   23     4         2
## 6  2018-11-25 02:00:00    2    11         0
## 7  2018-11-21 18:00:00   18    11         3
## 8  2018-07-19 22:00:00   22     7         4
## 9  2018-03-19 03:00:00    3     3         1
## 10 2018-12-21 15:00:00   15    12         5
```

Ptype

The ptype variable is grouped into simplified categories (none, rain, other) to reduce sparsity, then one-hot encoded to make it suitable for machine learning models

```r
library(dplyr)
library(fastDummies)

# Group and encode
train$ptype_grouped <- case_when(
  train$ptype == 0 ~ "none",
  train$ptype == 1 ~ "rain",
  TRUE             ~ "other"
)

train <- dummy_cols(
  train,
  select_columns = "ptype_grouped",
  remove_selected_columns = TRUE,
  remove_first_dummy = FALSE
)
```

Check new ptype distribution

```r
# Sum the one-hot columns to see category counts
ptype_distribution <- train %>%
  summarise(
    none  = sum(ptype_grouped_none),
    rain  = sum(ptype_grouped_rain),
    other = sum(ptype_grouped_other)
  )

print(ptype_distribution)
```

```
##      none    rain  other
```

```
## 1 4825273 8000308 463339
```

Wind transformation

```r
# Calculate wind speeds
train$wind10_speed  <- sqrt(train$u10^2 + train$v10^2)
train$wind100_speed <- sqrt(train$u100^2 + train$v100^2)
```

```r
# Wind direction (0° = North, 90° = East)
train$wind10_dir  <- (atan2(train$u10,  train$v10)  * 180 / pi) %% 360
train$wind100_dir <- (atan2(train$u100, train$v100) * 180 / pi) %% 360

# Convert to radians
train$wind10_dir_rad  <- train$wind10_dir  * pi / 180
train$wind100_dir_rad <- train$wind100_dir * pi / 180

# Sine and cosine components
train$wind10_dir_sin  <- sin(train$wind10_dir_rad)
train$wind10_dir_cos  <- cos(train$wind10_dir_rad)

train$wind100_dir_sin <- sin(train$wind100_dir_rad)
train$wind100_dir_cos <- cos(train$wind100_dir_rad)
```

Standarization

Selected numeric features are standardized to have a mean of 0 and standard deviation of 1. This ensures that features on different scales contribute equally to the model, especially for distance-based or regularized algorithms

```r
# List of features to scale
features_to_scale <- c(
  "tp", "sp", "wind10_speed", "wind100_speed",
  "hour", "month", "dayofweek"
)

# Create a copy to preserve the original dataset
train_scaled <- train

# Apply standard scaling (mean = 0, sd = 1)
train_scaled[features_to_scale] <- scale(train_scaled[features_to_scale])

# Display scaled values to confirm
head(train_scaled[features_to_scale])
```

```
##           tp        sp wind10_speed wind100_speed      hour     month dayofweek
## 1 -0.3469539 -1.455549   -1.2034794    -1.4402091 -1.661325 -1.602745 -0.997264
## 2 -0.3069347 -1.484714   -1.0098109    -1.2894574 -1.661325 -1.602745 -0.997264
## 3 -0.3024881 -1.489575   -0.8790867    -1.1795353 -1.661325 -1.602745 -0.997264
## 4 -0.2995237 -1.490790   -0.8224914    -1.1231176 -1.661325 -1.602745 -0.997264
## 5 -0.3113813 -1.485929   -0.6925897    -1.0113682 -1.661325 -1.602745 -0.997264
## 6 -0.3410252 -1.487145   -0.5959104    -0.9391121 -1.661325 -1.602745 -0.997264
```

Time sin-cos transformation

Time features like hour, month, and dayofweek are cyclical (e.g., 23:00 is close to 00:00), so we transform them using sine and cosine functions. This captures their natural circular patterns and allows the model to learn transitions more smoothly (e.g., midnight to early morning). Without this encoding, the model might incorrectly interpret 0 and 23 as being far apart.

```r
# Encode 'hour' as cyclical
train_scaled$hour_sin <- sin(2 * pi * train_scaled$hour / 24)
train_scaled$hour_cos <- cos(2 * pi * train_scaled$hour / 24)

# Encode 'month' as cyclical
train_scaled$month_sin <- sin(2 * pi * train_scaled$month / 12)
train_scaled$month_cos <- cos(2 * pi * train_scaled$month / 12)

# Encode 'dayofweek' as cyclical (0 = Monday, 6 = Sunday)
train_scaled$dow_sin <- sin(2 * pi * train_scaled$dayofweek / 7)
train_scaled$dow_cos <- cos(2 * pi * train_scaled$dayofweek / 7)

library(dplyr)

train_scaled %>%
  select(hour, hour_sin, hour_cos, month, month_sin, month_cos) %>%
  sample_n(10)
```

```
##          hour    hour_sin  hour_cos      month   month_sin month_cos
## 1  -1.5168617 -0.38675805 0.9221812  0.7175404  0.36692683 0.9302498
## 2   0.2166945  0.05670007 0.9983913  1.0075761  0.50343142 0.8640352
## 3   0.3611575  0.09441001 0.9955334  1.5876475  0.73880153 0.6739231
## 4   1.3723987  0.35161259 0.9361456 -1.3127095 -0.63447808 0.7729409
## 5  -1.0834726 -0.27986403 0.9600396  0.1374690  0.07191645 0.9974107
## 6   1.5168617  0.38675805 0.9221812 -1.3127095 -0.63447808 0.7729409
## 7   1.6613247  0.42135036 0.9068979  0.1374690  0.07191645 0.9974107
## 8   1.2279357  0.31596426 0.9487711  1.2976118  0.62834810 0.7779323
## 9   0.2166945  0.05670007 0.9983913  1.0075761  0.50343142 0.8640352
## 10  1.3723987  0.35161259 0.9361456 -1.6027452 -0.74410586 0.6680617
```

x and y

```r
# Define the target variable
y <- train_scaled$t2m

# Define input features
feature_cols <- c(
  "tp", "sp", "wind10_speed", "wind100_speed",
  "hour_sin", "hour_cos",
  "month_sin", "month_cos",
  "wind10_dir_sin", "wind10_dir_cos",
  "ptype_grouped_none", "ptype_grouped_rain", "ptype_grouped_other"
)

# Create feature matrix
X <- train_scaled[, feature_cols]

# Print shape
```

```r
cat(" Features and target selected.\n")
```

```
##  Features and target selected.
```

```r
cat("Feature matrix dimensions:", nrow(X), "rows ×", ncol(X), "columns\n")
```

```
## Feature matrix dimensions: 13288920 rows × 13 columns
```

4% sample, and 80-20 split

```r
library(dplyr)

set.seed(42)
train_scaled %>% sample_n(5)
```

```
##            valid_time latitude longitude          tp        u10        v10
## 1 2018-03-30 02:00:00    58.00      1.50 -0.03124643 -11.239471  4.5898895
## 2 2018-04-27 16:00:00    52.25     -2.75  0.18070746  -3.664642 -2.8932037
## 3 2018-05-13 08:00:00    58.25      0.00 -0.35881149  -9.211166  3.0352020
## 4 2018-09-19 05:00:00    50.25      1.25 -0.35881149   2.761581  6.7854156
## 5 2018-03-29 16:00:00    58.00     -7.25 -0.35881149  -3.684509 -0.3172607
##           sp       u100       v100       tcc ptype      t2m       hour
## 1  0.1729806 -12.920395  5.4011080 0.9953308     1 276.5325 -1.3723987
## 2 -1.1548218  -5.393127 -4.0372010 1.0000000     1 280.0032  0.6500836
## 3  0.5574814 -12.542831  5.0771637 0.8540649     0 283.0137 -0.5056206
## 4  0.5872145   4.450867  8.5746765 0.8445129     0 290.6467 -0.9390096
## 5 -0.5242811  -4.636353 -0.2922669 0.4179993     0 280.9066  0.6500836
##        month    dayofweek ptype_grouped_none ptype_grouped_other
## 1 -1.0226738  1.002743452                  0                   0
## 2 -0.7326381  1.002743452                  0                   0
## 3 -0.4426024 -1.497265837                  1                   0
## 4  0.7175404  0.002739736                  1                   0
## 5 -1.0226738  0.502741594                  1                   0
##   ptype_grouped_rain wind10_speed wind100_speed wind10_dir wind100_dir
## 1                  1    1.6383258     1.2685709  292.21375   292.68637
## 2                  1   -0.4475623    -0.3865472  231.70921   233.18212
## 3                  0    0.9565139     1.1609731  288.23774   292.03742
## 4                  0    0.2941602     0.2794569   22.14572    27.43252
## 5                  0   -0.7186265    -0.8628474  265.07859   266.39296
##   wind10_dir_rad wind100_dir_rad wind10_dir_sin wind10_dir_cos wind100_dir_sin
## 1      5.1000920       5.1083409     -0.9257799      0.3780629      -0.9226298
## 2      4.0440886       4.0697957     -0.7848760     -0.6196529      -0.8005443
## 3      5.0306976       5.0970145     -0.9497661      0.3129606      -0.9269390
## 4      0.3865158       0.4787878      0.3769635      0.9262281       0.4607036
## 5      4.6264942       4.6494342     -0.9963133     -0.0857892      -0.9980190
##   wind100_dir_cos   hour_sin  hour_cos  month_sin month_cos      dow_sin
## 1      0.38568662 -0.3516126 0.9361456 -0.5102460 0.8600285  0.783364464
## 2     -0.59927352  0.1693711 0.9855524 -0.3742691 0.9273202  0.783364464
## 3      0.37521203 -0.1319849 0.9912517 -0.2296773 0.9732668 -0.974378871
## 4      0.88755403 -0.2433635 0.9699351  0.3669268 0.9302498  0.002459179
## 5     -0.06291322  0.1693711 0.9855524 -0.5102460 0.8600285  0.436099571
##     dow_cos
## 1 0.6215626
## 2 0.6215626
```

```
## 3 0.2249129
## 4 0.9999970
## 5 0.8998984
```

```r
set.seed(99)

# Step 1: Sample 4% of the full dataset
train_sample <- train_scaled[sample(nrow(train_scaled), size = 0.04 * nrow(train_scaled)), ]

# Step 2: Split into 80% training and 20% validation
train_index <- sample(seq_len(nrow(train_sample)), size = 0.8 * nrow(train_sample))

train_subset <- train_sample[train_index, ]
val_subset   <- train_sample[-train_index, ]

# Confirm dimensions
cat(" Sampled dataset size:", nrow(train_sample), "\n")
```

```
##  Sampled dataset size: 531556
```

```r
cat(" Training set size   :", nrow(train_subset), "\n")
```

```
##  Training set size   : 425244
```

```r
cat(" Validation set size :", nrow(val_subset), "\n")
```

```
##  Validation set size : 106312
```

Random forest

A Random Forest model is trained using 4% of the dataset with 500 trees. The model uses 7 features at each split (mtry = 7) and a minimum node size of 10. Predictions are made on a held-out validation set, and performance is evaluated using RMSE and MAE. This model balances predictive power with interpretability and is well-suited to handle non-linear relationships in the data

```r
# Load the ranger package
library(ranger)

# Fit the Random Forest model
rf_model <- ranger(
  formula = t2m ~ .,
  data = train_subset,
  num.trees = 500,
  mtry = 7,
  min.node.size = 10,
  importance = "impurity",
  seed = 99
)

# Predict on the validation set
y_val <- val_subset$t2m
X_val <- val_subset[, !(names(val_subset) %in% c("t2m"))]

y_pred <- predict(rf_model, data = X_val)$predictions

# Evaluate
rmse <- sqrt(mean((y_val - y_pred)^2))
```

```r
mae  <- mean(abs(y_val - y_pred))

# Print results
cat(" Random Forest Evaluation (4% Sample):\n")
cat("RMSE:", round(rmse, 3), "\n")
cat("MAE :", round(mae, 3), "\n")
```

RMSE: 0.876 MAE : 0.6 (R crashes when knitting)

```r
library(ranger)

# Fit the model with a tuned configuration
rf_model <- ranger(
  formula = t2m ~ .,
  data = train_subset,
  num.trees = 800,        # More trees for better generalization
  mtry = 9,               # Slightly higher than best so far
  min.node.size = 5,      # Smaller leaves for more complexity
  importance = "impurity",
  seed = 99
)

# Predict and evaluate
y_val <- val_subset$t2m
X_val <- val_subset[, !(names(val_subset) %in% "t2m")]

y_pred <- predict(rf_model, data = X_val)$predictions

# Metrics
rmse <- sqrt(mean((y_val - y_pred)^2))
mae  <- mean(abs(y_val - y_pred))

# Output
cat(" Tuned Random Forest:\n")
cat("RMSE:", round(rmse, 3), "\n")
cat("MAE :", round(mae, 3), "\n")
```

RMSE: 0.817 MAE : 0.554 (R crashed when knitting)