AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

408/1, Kuratoli, Khilkhet, Dhaka 1229, Bangladesh

Project Title: Apply Naïve Bayes Classification Algorithm

Project No: Final

Date of Submission: 25/12/23

Course Title: INTRODUCTION TO DATA SCIENCE

Course Code: 00489

Section: B

Semester: Fall

Course Teacher: TOHEDUL ISLAM

No	Name	ID	Program	Signature
1	**MD. AJMAIN FAIEQ**	21-45192-2	BSc CSE	ajmain
2	**RAKIB AHAMED LIMON**	20-42900-1	BSc CSE	limon

Faculty use only		
FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

Data Set Notes:

The dataset appears to be comprehensive, including a variety of weather-related parameters. It useful for weather analysis, prediction models, or studying climatic patterns in specific locations. Some columns have missing data, which is common in real-world datasets and might require handling during analysis**.**

Here's a brief overview based on the first few rows:

- Date: The date of the weather observation.
- Location: The place where the weather data was recorded.
- MinTemp and MaxTemp: The minimum and maximum temperatures recorded on that day.
- Rainfall: The amount of rainfall.
- Evaporation: The amount of evaporation. (Some values are missing in this column).
- Sunshine: The amount of sunshine. (This column also contains missing values).
- WindGustDir and WindGustSpeed: The direction and speed of the strongest wind gusts.
- WindDir9am and WindDir3pm: The wind direction at 9 am and 3 pm.
- WindSpeed9am and WindSpeed3pm: The wind speed at 9 am and 3 pm.
- Humidity9am and Humidity3pm: The humidity percentages at 9 am and 3 pm.
- Pressure9am and Pressure3pm: The atmospheric pressure at 9 am and 3 pm.
- Cloud9am and Cloud3pm: The amount of cloud cover at 9 am and 3 pm. (This column also has some missing values).
- Temp9am and Temp3pm: The temperature at 9 am and 3 pm.
- RainToday: Indicates whether it rained that day.
- RainTomorrow: Indicates whether it will rain the next day.

The Project:

Library used:

```
library(lubridate)

library(dplyr)

library(ggplot2)

library(reshape2)

library(e1071)

library(caret)

library(caTools)
```

library(readr)

data <- read.csv("train_mod1.csv")
View(data)
summary(data)

- The read.csv() function is used to read CSV files in R.
- View(data) function to open a data viewer window. This window will display the contents of the "data" variable, allowing you to explore the data interactively.
- The summary() function is used to generate summary statistics and information about the data.

missing_values <- colSums(is.na(data))
print("Missing Values:")
print(missing_values)

empty_string <- colSums(data == "" | data == " ")
print("Empty string:")
print(empty_string)

unique_counts <- sapply(data, function(x) length(unique(x)))
unique_counts

- Here we count missing values, empty strings and unique values

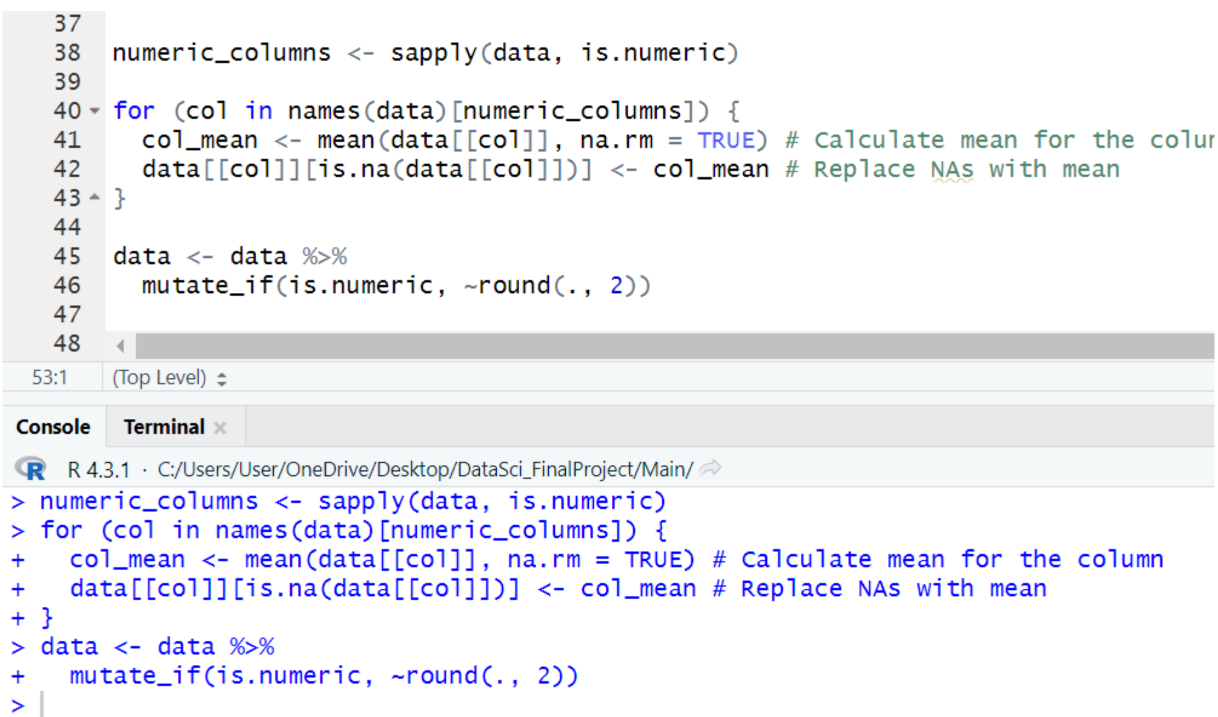
```
> # Count missing values in each column
> missing_values <- colSums(is.na(data))
> print("Missing Values:")
[1] "Missing Values:"
> print(missing_values)
         Date      Location       MinTemp       MaxTemp      Rainfall   Evaporation 
            0             0            12            13            41           769 
     Sunshine   WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am 
          851           121           120           118            53            19 
 WindSpeed3pm   Humidity9am   Humidity3pm   Pressure9am   Pressure3pm      Cloud9am 
           34            32            54           162           157           684 
     Cloud3pm       Temp9am       Temp3pm     RainToday  RainTomorrow 
          727            15            37            41            42 
> # Count empty strings in each column
> empty_string <- colSums(data == "" | data == " ")
> print("Empty string:")
[1] "Empty string:"
> print(empty_string)
         Date      Location       MinTemp       MaxTemp      Rainfall   Evaporation 
            0             0            NA            NA            NA            NA 
     Sunshine   WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am 
           NA            NA            NA            NA            NA            NA 
 WindSpeed3pm   Humidity9am   Humidity3pm   Pressure9am   Pressure3pm      Cloud9am 
           NA            NA            NA            NA            NA            NA 
     Cloud3pm       Temp9am       Temp3pm     RainToday  RainTomorrow 
           NA            NA            NA            NA            NA 
```

numeric_columns <- sapply(data, is.numeric)

for (col in names(data)[numeric_columns]) {
col_mean <- mean(data[[col]], na.rm = TRUE) # Calculate mean for the column
data[[col]][is.na(data[[col]])] <- col_mean # Replace NAs with mean
}

data <- data %>%
mutate_if(is.numeric, ~round(., 2))

- numeric_columns <- sapply(data, is.numeric) is used to identify which columns in the "data" dataframe are numeric. It creates a logical vector.
- A **for** loop is used to iterate over the names of the columns in the "data" dataframe that are identified as numeric in the previous step. Inside the loop, for each numeric column (**col**), the code calculates the mean of that column's values using the **mean()** function. The **na.rm = TRUE** argument ensures that any missing values (NAs) are ignored when calculating the mean.
- After replacing missing values, the code uses the **mutate_if()** function to round all numeric columns in the "data" dataframe to two decimal places. The **is.numeric** function is used to identify numeric columns, and the **~round(., 2)** formula is applied to round each numeric column.



Co-Relation Check (**Pearson's Chi-squared test):**

```
data$RainTomorrow <- as.factor(data$RainTomorrow)
p_values <- vector()
for (col in names(data)[-which(names(data) == "RainTomorrow")]) {
  data[[col]] <- as.factor(data[[col]])

  test_result <- chisq.test(table(data[[col]], data$RainTomorrow))
  p_values[col] <- test_result$p.value
}
print(p_values)
significant_attributes <- names(p_values)[p_values < 0.05]
print(significant_attributes)
```

This code performs a chi-squared test for each attribute in the dataset (excluding the target variable) to determine their statistical significance in predicting whether it will rain tomorrow.

- **Converting the Target Variable to a Factor:**
    - The code begins by converting the "RainTomorrow" column in the dataset to a factor variable. This is often done when the target variable is categorical, and it's necessary for statistical tests.
- **Initializing an Empty Vector for p-values:**
    - A vector called "p_values" is initialized to store p-values from a chi-squared test. This vector will later store the p-values for each attribute in the dataset when compared to the "RainTomorrow" variable.
- **Iterating Over Columns (Except "RainTomorrow"):**
    - A **for** loop iterates over all column names in the dataset except for "RainTomorrow" (the target variable).
- **Converting Each Column to a Factor:**
    - Inside the loop, each column (attribute) in the dataset (except "RainTomorrow") is converted to a factor variable using the **as.factor()** function. This step is necessary for the chi-squared test, which requires categorical variables.
- **Performing a Chi-Squared Test:**
    - For each attribute, a chi-squared test is performed to assess its independence with respect to the "RainTomorrow" variable. The **table()** function is used to create a contingency table of the two variables, and **chisq.test()** is applied to compute the chi-squared test statistics.
- **Storing p-values:**
    - The p-value from each chi-squared test is stored in the "p_values" vector, with the index corresponding to the column being tested.
- **Printing p-values:**

- The code prints the p-values for each attribute, showing how statistically significant each attribute is in relation to "RainTomorrow."

- **Identifying Significant Attributes:**
  - Finally, the code identifies the attributes with p-values less than 0.05 (common significance threshold) and stores their names in the "significant_attributes" vector. These are the attributes that are considered statistically significant in relation to the "RainTomorrow" variable.

```
> print(p_values)
         Date     Location      MinTemp      MaxTemp     Rainfall  Evaporation 
 6.929975e-01 2.097276e-07 4.124609e-01          NaN          NaN 4.030200e-02 
     Sunshine  WindGustDir WindGustSpeed   WindDir9am   WindDir3pm WindSpeed9am 
          NaN 3.086442e-01 3.174377e-17 1.078794e-04 3.480379e-02 7.518586e-02 
 WindSpeed3pm  Humidity9am  Humidity3pm  Pressure9am  Pressure3pm     Cloud9am 
 3.763519e-03 3.476348e-09 1.254006e-64          NaN          NaN 2.763748e-26 
     Cloud3pm      Temp9am      Temp3pm    RainToday 
 3.080553e-41          NaN          NaN 9.892852e-49 
> significant_attributes <- names(p_values)[p_values < 0.05]
> print(significant_attributes)
 [1] "Location"      NA              NA              "Evaporation"   NA             
 [6] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed3pm"  "Humidity9am"  
[11] "Humidity3pm"   NA              NA              "Cloud9am"      "Cloud3pm"     
[16] NA              NA              "RainToday"    
> 
```

```
 for(col in names(data)){
  if(is.factor(data[[col]]) || is.character(data[[col]])){
   data[[col]] <- as.integer(factor(data[[col]]))
  }
 }
get_mode <- function(v) {
 uniqv <- unique(v)
 uniqv[which.max(tabulate(match(v, uniqv)))]
}
columns_to_replace <- c("WindGustDir", "WindDir9am", "WindDir3pm", "RainToday",
"RainTomorrow")
for (col in columns_to_replace) {
 mode_value <- get_mode(data[[col]][!is.na(data[[col]])])
 data[[col]][is.na(data[[col]])] <- mode_value
}
```

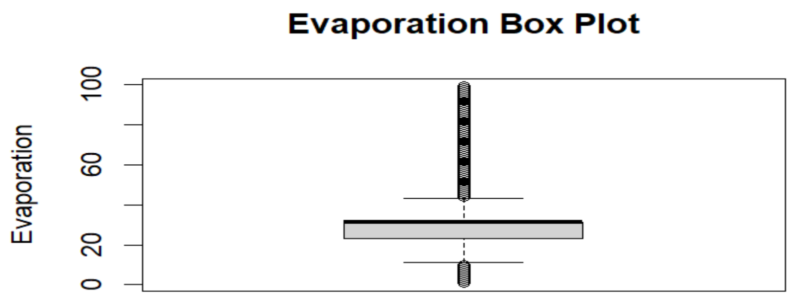
- **Converting Factor and Character Columns to Integer**:
  - The code iterates over all columns in the "data" dataframe.
  - For each column, it checks if the column is either a factor or a character column using the is.factor() and is.character() functions.

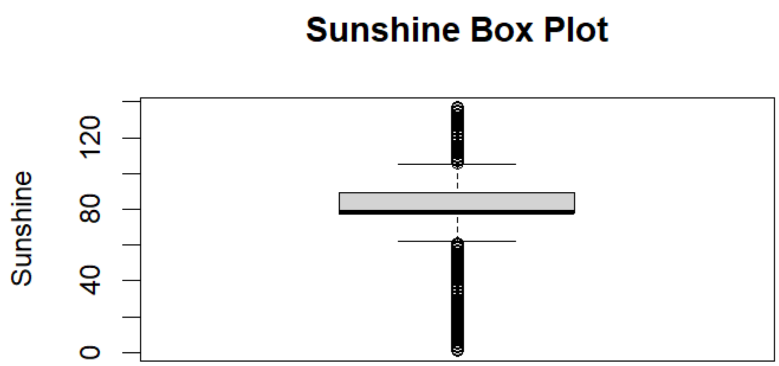
  - If the column is a factor or character, it is converted to an integer using as.integer(factor(data[[col]])). This can be useful for machine learning algorithms that require numeric input.
- **Defining a Custom Function to Get Mode**:
  - The code defines a custom function called get_mode(v) which calculates the mode (most frequently occurring value) of a vector v. It does this by finding the unique values, counting their occurrences, and returning the one with the highest count.
- **Specifying Columns to Replace Missing Values**:
  - A vector called "columns_to_replace" is defined, which contains the names of columns in the dataset that need to have missing values replaced. These columns include "WindGustDir," "WindDir9am," "WindDir3pm," "RainToday," and "RainTomorrow."
- **Replacing Missing Values with Mode**:
  - A for loop iterates over the columns specified in "columns_to_replace."
  - For each column, it calculates the mode (most frequent value) of that column, excluding missing values (!is.na(data[[col]])).
  - It then replaces missing values (NAs) in that column with the calculated mode value.

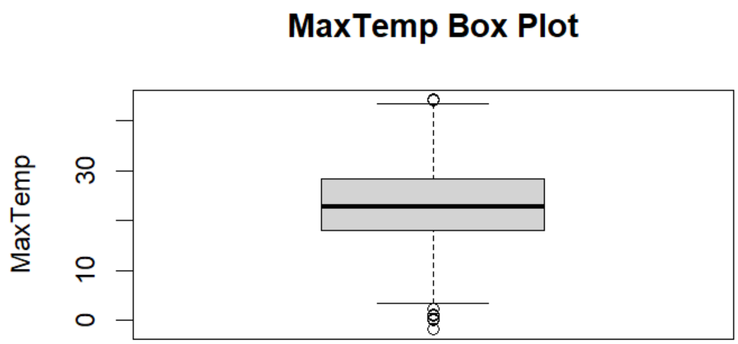
```
num_col <- sapply(data, is.numeric)
for (col in names(data)[num_col]) {
  boxplot(data[[col]], main = paste(col, "Box Plot"), ylab = col)
}
```

- **Identifying Numeric Columns**:
  - The code starts by using sapply() to create a logical vector called "num_col." Each element of this vector corresponds to a column in the "data" dataframe and indicates whether the column contains numeric data (e.g., numbers).
- **Creating Box Plots for Numeric Columns**:
  - A for loop is used to iterate over the names of the columns in the "data" dataframe that are identified as numeric in the previous step (names(data)[num_col]).
  - For each numeric column (col), a box plot is created using the boxplot() function.
- **Customizing the Box Plot**:
  - The boxplot() function is configured with several options:
    - data[[col]]: This specifies the numeric data to be plotted.
    - main = paste(col, "Box Plot"): The title of the box plot is set to include the name of the column, creating a title like "ColumnName Box Plot."
    - ylab = col: The y-axis label is set to match the name of the column, so it indicates what the data represents.

- **Displaying Box Plots**:
    - As the loop iterates through each numeric column, it generates a box plot for that column with a title and y-axis label specific to that column.








```
evaporation_column <- data$Evaporation

replace_outliers_with_mean <- function(x) {
  Q1 <- quantile(x, 0.31, na.rm = TRUE)
  Q3 <- quantile(x, 0.69, na.rm = TRUE)
  IQR <- Q3 - Q1

  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR

  mean_value <- mean(x[x >= lower_bound & x <= upper_bound], na.rm = TRUE)

  x[x < lower_bound | x > upper_bound] <- mean_value
  return(x)
}
data$Evaporation <- replace_outliers_with_mean(evaporation_column)
```

```
MaxTemp_column <- data$MaxTemp
replace_outliers_with_mean <- function(x) {
  Q1 <- quantile(x, 0.3, na.rm = TRUE)
  Q3 <- quantile(x, 0.7, na.rm = TRUE)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  mean_value <- mean(x[x >= lower_bound & x <= upper_bound], na.rm = TRUE)
  x[x < lower_bound | x > upper_bound] <- mean_value
  return(x)
```

```
}
data$MaxTemp <- replace_outliers_with_mean(MaxTemp_column)

Sunshine_column <- data$Sunshine
replace_outliers_with_mean <- function(x) {
 Q1 <- quantile(x, 0.3, na.rm = TRUE)
 Q3 <- quantile(x, 0.7, na.rm = TRUE)
 IQR <- Q3 - Q1
 lower_bound <- Q1 - 1.5 * IQR
 upper_bound <- Q3 + 1.5 * IQR
 mean_value <- mean(x[x >= lower_bound & x <= upper_bound], na.rm = TRUE)
 x[x < lower_bound | x > upper_bound] <- mean_value
 return(x)
}
data$Sunshine <- replace_outliers_with_mean(Sunshine_column)
```

- Replace outliers with mean value for Evaporation, MaxTemp and Sunshine column.

Naïve Bayes and 10-fold Cross validation:

```
set.seed(123)  # for reproducibility
split <- sample.split(data$RainTomorrow, SplitRatio = 0.7)
train_data <- subset(data, split == TRUE)
test_data <- subset(data, split == FALSE)
```

- **Data Splitting for Training and Testing**:
  - sample.split(data$RainTomorrow, SplitRatio = 0.7) is a function that splits the dataset into two subsets, typically for the purpose of training and testing a machine learning model.
  - The data$RainTomorrow part specifies that the target variable for splitting is "RainTomorrow."
  - SplitRatio = 0.7 indicates that approximately 70% of the data will be used for training, and the remaining 30% will be used for testing.
- **Creating Training and Testing Datasets**:
  - train_data <- subset(data, split == TRUE) creates a new dataframe called "train_data" by subsetting the original "data" dataframe. It selects rows where the "split" condition is TRUE, which corresponds to the training set based on the earlier splitting process.
  - test_data <- subset(data, split == FALSE) creates a new dataframe called "test_data" by subsetting the original "data" dataframe. It selects rows where the "split" condition is FALSE, which corresponds to the testing set.

```
train_data$RainTomorrow <- as.factor(train_data$RainTomorrow)
```

```
test_data$RainTomorrow <- as.factor(test_data$RainTomorrow)
control <- trainControl(method = "cv", number = 10)
model <- train(RainTomorrow~ ., data = train_data, method = "naive_bayes", trControl =
control)
predictions <- predict(model, test_data)
confusionMatrix <- confusionMatrix(predictions, test_data$RainTomorrow)
recall <- confusionMatrix$byClass["Sensitivity"]
precision <- confusionMatrix$byClass["Pos Pred Value"]
F1 <- 2 * (precision * recall) / (precision + recall)
accuracy <- sum(predictions == test_data$RainTomorrow) / nrow(test_data)
print(paste("Accuracy on Test Set:", accuracy))
```

- Converting Target Variables to Factors:
    - The code converts the "RainTomorrow" column in both the training and testing datasets to a factor variable. This is often done when the target variable is categorical.
- Setting Up Cross-Validation Control:
    - The trainControl() function is used to set up control parameters for model training. It specifies a 10-fold cross-validation (method = "cv"), which will be used during the training process.
- Training a Naive Bayes Model:
    - The train() function is used to train a classification model. It predicts "RainTomorrow" based on all other variables in the training dataset (RainTomorrow~ .).
    - The method chosen for modeling is "naive_bayes," indicating that a Naive Bayes classifier will be used.
    - The cross-validation control parameters are specified with trControl = control.
- Making Predictions:
    - The model is used to make predictions on the testing dataset using the predict() function. Predicted values are stored in the "predictions" variable.
- Calculating Evaluation Metrics:
    - The code calculates various evaluation metrics for the model's performance on the test dataset:
        - Recall (Sensitivity): Measures the proportion of actual positives that were correctly predicted as positives.
        - Precision (Pos Pred Value): Measures the proportion of predicted positives that were actually positive.
        - F1 Score: A harmonic mean of precision and recall, providing a single metric that balances both.
        - Accuracy: Calculates the overall accuracy of the model by comparing predicted values to the actual values in the test dataset.
- Printing the Accuracy:
    - The code prints the accuracy of the model on the test set.

```r
train_data$RainTomorrow <- as.factor(train_data$RainTomorrow)
test_data$RainTomorrow <- as.factor(test_data$RainTomorrow)

control <- trainControl(method = "cv", number = 10)
model <- train(RainTomorrow~ ., data = train_data, method = "naive_bayes", trControl =

predictions <- predict(model, test_data)
confusionMatrix <- confusionMatrix(predictions, test_data$RainTomorrow)

recall <- confusionMatrix$byClass["Sensitivity"]
precision <- confusionMatrix$byClass["Pos Pred Value"]
F1 <- 2 * (precision * recall) / (precision + recall)

# Calculate accuracy
accuracy <- sum(predictions == test_data$RainTomorrow) / nrow(test_data)
print(paste("Accuracy on Test Set:", accuracy))
```

162:1 (Top Level) R Scrip

Console Terminal

R 4.3.1 · C:/Users/User/OneDrive/Desktop/DataSci_FinalProject/Main/

```
>
> recall <- confusionMatrix$byClass["Sensitivity"]
> precision <- confusionMatrix$byClass["Pos Pred Value"]
> F1 <- 2 * (precision * recall) / (precision + recall)
>
> # Calculate accuracy
> accuracy <- sum(predictions == test_data$RainTomorrow) / nrow(test_data)
> print(paste("Accuracy on Test Set:", accuracy))
[1] "Accuracy on Test Set: 0.809259259259259"
>
```

print(paste("Recall:", recall))
print(paste("Precision:", precision))
print(paste("F1 Score:", F1))
print(accuracy)
print(confusionMatrix)

```
> print(paste("Recall:", recall))
[1] "Recall: 0.892271662763466"
> print(paste("Precision:", precision))
[1] "Precision: 0.86986301369863"
> print(paste("F1 Score:", F1))
[1] "F1 Score: 0.880924855491329"
> print(accuracy)
[1] 0.8092593
> print(confusionMatrix)
Confusion Matrix and Statistics

          Reference
Prediction   1   2
         1 381  57
         2  46  56

               Accuracy : 0.8093
                 95% CI : (0.7735, 0.8416)
    No Information Rate : 0.7907
    P-Value [Acc > NIR] : 0.1574

                  Kappa : 0.4022

 Mcnemar's Test P-Value : 0.3245

            Sensitivity : 0.8923
            Specificity : 0.4956
         Pos Pred Value : 0.8699
         Neg Pred Value : 0.5490
             Prevalence : 0.7907
         Detection Rate : 0.7056
   Detection Prevalence : 0.8111
```

```
confusionMatrix <- confusionMatrix(predictions, test_data$RainTomorrow)
confusionMatrixTable <- as.table(confusionMatrix$table)
ggplot(data = as.data.frame(confusionMatrixTable), aes(x = Reference, y = Prediction)) +
  geom_tile(aes(fill = Freq), colour = "black") +
  geom_text(aes(label = sprintf("%0.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "yellow", high = "steelblue") +
  theme_minimal() +
  labs(fill = "Count")
```