Predicting the Weather using Machine Learning with R

Myron Keith Gibert Jr

2022-11-02

Contents

Introduction

Assignment Scenario

Congratulations! You have just been hired by a US Weather forecast firm as a data scientist.

The company is considering the weather condition to help predict the possibility of precipitations, which involves using various local climatological variables, including temperature, wind speed, humidity, dew point, and pressure. The data you will be handling was collected by a NOAA weather station located at the John F. Kennedy International Airport in Queens, New York.

Your task is to provide a high level analysis of weather data in JFK Airport. Your stakeholders want to understand the current and historical record of precipitations based on different variables. For now they are mainly interested in a macro-view of JFK Airport Weather, and how it relates to the possibility to rain because it will affect flight delays and etc.

The Data

This project relates to the NOAA Weather Dataset - JFK Airport (New York). The original dataset contains 114,546 hourly observations of 12 local climatological variables (such as temperature and wind speed) collected at JFK airport. This dataset can be obtained for free from the IBM Developer Data Asset Exchange.

For this project, you will be using a subset dataset, which contains 5727 rows (about 5% or original rows) and 9 columns. The end goal will be to predict the precipitation using some of the available features. In this project, you will practice reading data files, preprocessing data, creating models, improving models and evaluating them to ultimately choose the best model.

Import required modules

Below, install "tidymodels", additionally "rlang" should be updated in order to properly run "tidymodels".

```
knitr::opts_chunk$set(echo = TRUE)

if (!require("tidyverse")) install.packages("tidyverse")
library("tidyverse")

if (!require("ggplot2")) install.packages("ggplot2")
library("ggplot2")

if (!require("ggthemes")) install.packages("ggthemes")
library("ggthemes")

if (!require("rlang")) install.packages("rlang")
library("rlang")

if (!require("tidymodels")) install.packages("tidymodels")
library("tidymodels")

if (!require("glmnet")) install.packages("glmnet")
library("glmnet")
```

Understand the Dataset

Understand the Dataset The original NOAA JFK dataset contains 114,546 hourly observations of various local climatological variables (including temperature, wind speed, humidity, dew point, and pressure).

In this project you will use a sample dataset, which is around 293 KB:

Link to the sample dataset

The sample contains 5727 rows (about 5% or original rows) and 9 columns, which are:

- DATE
- HOURLYDewPointTempF
- HOURLYRelativeHumidity
- HOURLYDRYBULBTEMPF
- HOURLYWETBULBTEMPF
- HOURLYPrecip
- HOURLYWindSpeed
- HOURLYSeaLevelPressure
- HOURLYStationPressure

The original dataset is much bigger. Feel free to explore the original dataset:

Link to the original dataset

For more information about the dataset, checkout the preview of NOAA Weather - JFK Airport:

Link to the preview

1. Download NOAA Weather Dataset

Use the download.file() function to download the sample dataset from the URL below. Then untar it.

URL = https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weather-data-jfk-airport/1.1.4/noaa-weather-sample-data.tar.gz

```
download.file(url = "https://dax-cdn.cdn.appdomain.cloud/
              dax-noaa-weather-data-jfk-airport/
              1.1.4/noaa-weather-sample-data.tar.gz"
              ,destfile = "noaa-weather-sample-data.tar.gz")

untar("noaa-weather-sample-data.tar.gz")
```

2. Extract and Read into Project

We start by reading in the raw dataset. You should specify the file name as "noaa-weather-sample-data/jfk_weather_sample.csv".

Then, display the first few rows, and use glimpse to confirm its integrity (5727 rows x 9 columns)

```
data <- read.csv("noaa-weather-sample-data/jfk_weather_sample.csv")

head(data)

glimpse(data)
```

```
## Rows: 5,727
## Columns: 9
## $ DATE                  <chr> "2015-07-25T13:51:00Z", "2016-11-18T23:51:00Z",~
## $ HOURLYDewPointTempF   <chr> "60", "34", "33", "18", "27", "35", "4", "14", ~
## $ HOURLYRelativeHumidity <int> 46, 48, 89, 48, 61, 79, 51, 65, 90, 94, 79, 37,~
## $ HOURLYDRYBULBTEMPF    <int> 83, 53, 36, 36, 39, 41, 19, 24, 54, 73, 83, 44,~
## $ HOURLYWETBULBTEMPF    <int> 68, 44, 35, 30, 34, 38, 15, 21, 52, 72, 78, 35,~
## $ HOURLYPrecip          <chr> "0.00", "0.00", "0.00", "0.00", "T", "0.00", "0~
## $ HOURLYWindSpeed       <int> 13, 6, 13, 14, 11, 6, 0, 11, 11, 5, 21, 7, 17, ~
```

```
## $ HOURLYSeaLevelPressure <dbl> 30.01, 30.05, 30.14, 29.82, NA, 29.94, 30.42, 3~
## $ HOURLYStationPressure  <dbl> 29.99, 30.03, 30.12, 29.80, 30.50, 29.92, 30.40~
```

3. Select Subset of Columns

The end goal of this project will be to predict HOURLYprecip (precipitation) using a few other variables. Before you can do this, you first need to preprocess the dataset. Section 3 to section 6 focuses on preprocessing.

The first step in preprocessing is to select a subset of data columns and inspect the column types.

The key columns that we will explore in this project are:

- HOURLYRelativeHumidity
- HOURLYDRYBULBTEMPF
- HOURLYPrecip
- HOURLYWindSpeed
- HOURLYStationPressure

Data Glossary:

- 'HOURLYRelativeHumidity' is the relative humidity given to the nearest whole percentage.
- 'HOURLYDRYBULBTEMPF' is the dry-bulb temperature and is commonly used as the standard air temperature reported. It is given here in whole degrees Fahrenheit.
- 'HOURLYPrecip' is the amount of precipitation in inches to hundredths over the past hour. For certain automated stations, precipitation will be reported at sub-hourly intervals (e.g. every 15 or 20 minutes) as an accumulated amount of all precipitation within the preceding hour. A "T" indicates a trace amount of precipitation.
- 'HOURLYWindSpeed' is the speed of the wind at the time of observation given in miles per hour (mph).
- 'HOURLYStationPressure' is the atmospheric pressure observed at the station during the time of observation. Given in inches of Mercury (in Hg).

Select those five columns and store the modified dataframe as a new variable. Then, show the first ten rows.

```
data_subset <- data %>%
  dplyr::select(HOURLYRelativeHumidity,
                HOURLYDRYBULBTEMPF,
                HOURLYPrecip,
                HOURLYWindSpeed,
                HOURLYStationPressure)

head(data_subset,10)
```

4. Clean Up Columns

From the dataframe preview above, we can see that the column HOURLYPrecip - which is the hourly measure of precipitation levels - contains both NA and T values. T specifies trace amounts of precipitation (meaning essentially no precipitation), while NA means not available, and is used to denote missing values. Additionally, some values also have "s" at the end of them, indicating that the precipitation was snow.

Inspect the unique values present in the column HOURLYPrecip (with unique(dataframe$column)) to see these values.

```
unique(data_subset$HOURLYPrecip)
```

```
##  [1] "0.00"  "T"     "0.06"  NA      "0.03"  "0.02"  "0.08"  "0.01"  "0.07"
## [10] "0.16"  "0.09"  "0.22"  "0.02s" "0.24"  "0.18"  "0.05"  "0.04"  "0.09s"
## [19] "0.11"  "0.14"  "0.25"  "0.10"  "0.01s" "0.58"  "0.12"  "0.13"  "0.46"
```

```
## [28] "1.07"  "1.19"  "0.34"  "0.20"  "0.36s" "0.42"  "0.17"  "0.27"  "0.35"
## [37] "0.31"  "0.33"  "0.23"  "0.26"  "0.28"  "0.75"  "0.19"  "0.36"  "0.03s"
## [46] "0.07s" "0.54"  "0.59"  "0.21"
```

Having characters in values (like the "T" and "s" that you see in the unique values) will cause problems when you create a model because values for precipitation should be numerical. So you need to fix these values that have characters.

Now, for the column HOURLYPrecip:

1. Replace all the T values with "0.0" and
2. Remove "s" from values like "0.02s". In R, you can use the method str_remove(column, pattern = "s")$to remove the character "s" from the end of values. The$"" tells R to match to the end of values. The pattern is a regex pattern.

Look here for more information about regex and matching to strings in R.

Remember that you can use tidyverse's mutate() to update columns.

You can check your work by checking if unique values of HOURLYPrecip still contain any T or s. Store the modified dataframe as a new variable.

```
data_subset_clean <- data_subset %>%
  mutate(HOURLYPrecip = str_remove(HOURLYPrecip,pattern = "s$"),
         HOURLYPrecip = str_replace(HOURLYPrecip,"T","0.00"))

unique(data_subset_clean$HOURLYPrecip)
```

```
##  [1] "0.00" "0.06" NA     "0.03" "0.02" "0.08" "0.01" "0.07" "0.16" "0.09"
## [11] "0.22" "0.24" "0.18" "0.05" "0.04" "0.11" "0.14" "0.25" "0.10" "0.58"
## [21] "0.12" "0.13" "0.46" "1.07" "1.19" "0.34" "0.20" "0.36" "0.42" "0.17"
## [31] "0.27" "0.35" "0.31" "0.33" "0.23" "0.26" "0.28" "0.75" "0.19" "0.54"
## [41] "0.59" "0.21"
```

5. Convert Columns to Numerical Types

Now that you have removed the characters in the HOURLYPrecip column, you can safely covert the column to a numeric type.

First, check the types of the columns. You will notice that all are dbl (double or numeric) except for HOURLYPrecip, which is chr (character or string). Use the glimpse function from Tidyverse. Then, convert HOURLYPrecip to the numeric type and store the cleaned dataframe as a new variable. Lastly, confirm that all columns have the numeric type.

```
glimpse(data_subset_clean)

#or

sapply(data_subset_clean,class)

data_subset_clean$HOURLYPrecip <- as.numeric(data_subset_clean$HOURLYPrecip)

sapply(data_subset_clean,class)
```

```
## HOURLYRelativeHumidity     HOURLYDRYBULBTEMPF           HOURLYPrecip
##              "integer"              "integer"              "numeric"
##        HOURLYWindSpeed  HOURLYStationPressure
##              "integer"              "numeric"
```

6. Rename Columns

Let's rename the following columns as:

- 'HOURLYRelativeHumidity' to 'relative_humidity'
- 'HOURLYDRYBULBTEMPF' to 'dry_bulb_temp_f'
- 'HOURLYPrecip' to 'precip'
- 'HOURLYWindSpeed' to 'wind_speed' '* HOURLYStationPressure' to 'station_pressure'

You can use dplyr::rename(). Then, store the final dataframe as a new variable.

```
data_subset_rename <- data_subset_clean %>%
  dplyr::rename(relative_humidity = HOURLYRelativeHumidity,
                dry_bulb_temp_f = HOURLYDRYBULBTEMPF,
                precip = HOURLYPrecip,
                wind_speed = HOURLYWindSpeed,
                station_pressure = HOURLYStationPressure)
```

7. Exploratory Data Analysis

Now that you have finished preprocessing the dataset, you can can start exploring the columns more.

Splitting Data into Training and Testing Sets

First, split the data into a training and testing set. Splitting a dataset is done randomly, so to have reproducible results set the seed = 1234. Also, use 80% of the data for training.

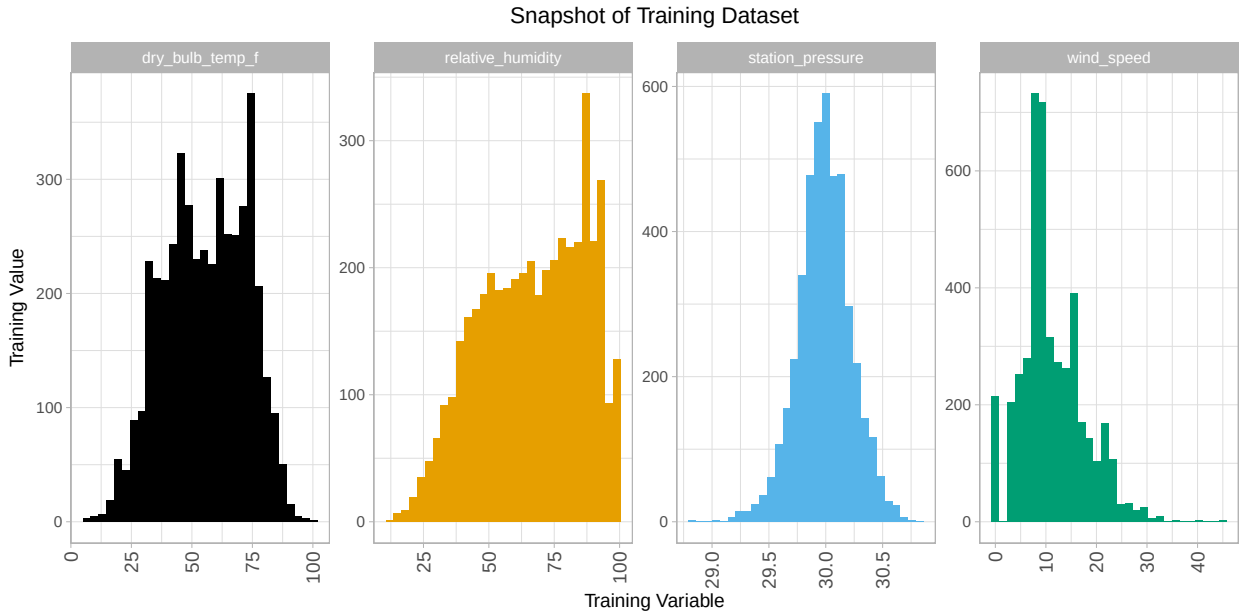
```
set.seed(1234)

data_subset_split <- initial_split(data_subset_rename,prop=0.80)
```

Taking a snapshot of the Training Dataset

Next, looking at just the training set, plot histograms or box plots of the variables (relative_humidity, dry_bulb_temp_f, precip, wind_speed, station_pressure) for an intial look of their distributions using tidyverse's ggplot. Leave the testing set as is because it is good practice to not see the testing set until evaluating the final model.

```
data_subset_training <- training(data_subset_split)

data_subset_training_plot <- data_subset_training %>%
  gather(key = "training_variable",value = "training_value",-precip)

p <- ggplot(data_subset_training_plot,aes(training_value,fill=training_variable)) +
  geom_histogram() +
  facet_wrap(~training_variable,scales="free",ncol=5) +
  xlab("Training Variable") +
  ylab(expression("Training Value")) +
  ggtitle(expression("Snapshot of Training Dataset")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_text(size=rel(1.2), face="bold",hjust = 0.5),
        legend.position = "none")
```

```
p
```



8. Designing a Simple Linear Regression model

After exploring the dataset more, you are now ready to start creating models to predict the precipitation (precip).

Create simple linear regression models where precip is the response variable and each of relative_humidity, dry_bulb_temp_f,wind_speed or station_pressure will be a predictor variable, e.g. precip ~ relative_humidity, precip ~ dry_bulb_temp_f, etc. for a total of four simple models. Additionally, visualize each simple model with a scatter plot.

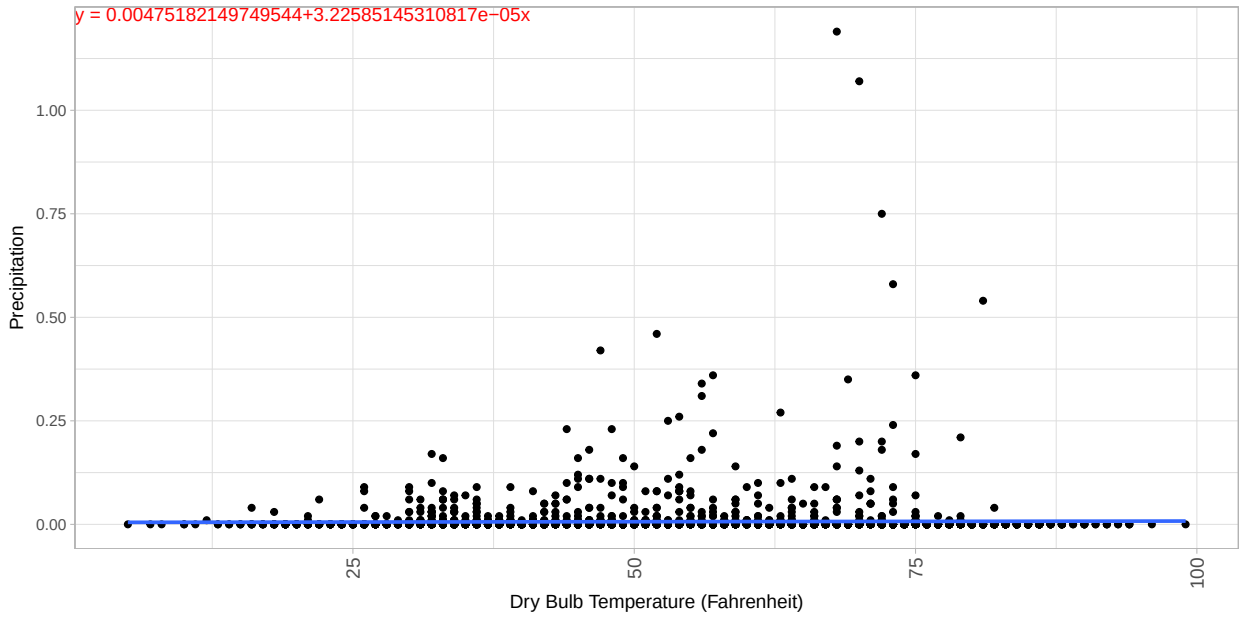
Design Model

```
lm_model <- linear_reg() %>%
            set_engine('lm') %>% # adds lm implementation of linear regression
            set_mode('regression')
```

Precipitation vs Dry Bulb Temperature (Fahrenheit)

```
lm_fit <- lm_model %>%
          fit(precip ~ dry_bulb_temp_f, data = data_subset_training)

fitdata <- tidy(lm_fit)

operator <- ifelse(as.numeric(fitdata$estimate[
  complete.cases(match(fitdata$term,"dry_bulb_temp_f"))])>=0,
  "+",
  "-")

reg_equation <- paste(
```

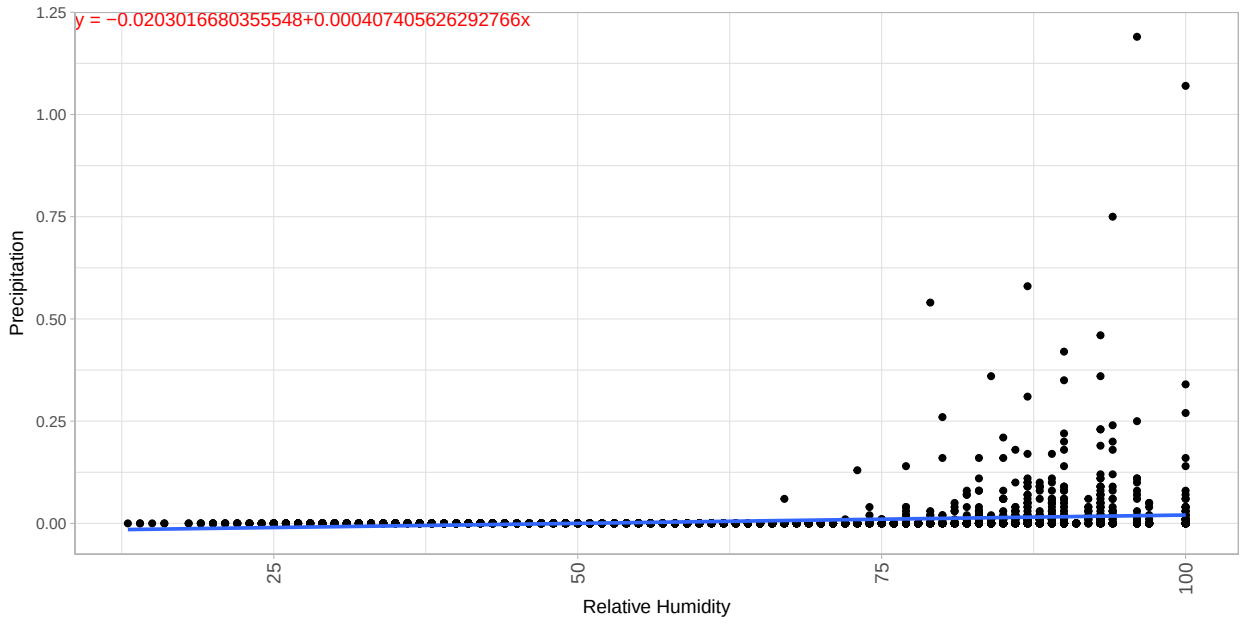
```
  "y = ",
  (fitdata$estimate[complete.cases(match(fitdata$term,"(Intercept)"))]),
  operator,
  abs(fitdata$estimate[complete.cases(match(fitdata$term,"dry_bulb_temp_f"))]),
  "x",
  sep="")

p <- ggplot(data_subset_training,(aes(dry_bulb_temp_f,precip))) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  xlab("Dry Bulb Temperature (Fahrenheit)") +
  ylab(expression("Precipitation")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_blank(),
        legend.position = "none") +
annotate('text', label=reg_equation, x=-Inf, y=Inf, hjust=0, vjust=1,color="red")


p
```



Precipitation vs Relative Humidity

```
lm_fit <- lm_model %>%
          fit(precip ~ relative_humidity, data = data_subset_training)

fitdata <- tidy(lm_fit)

operator <- ifelse(as.numeric(fitdata$estimate[
  complete.cases(match(fitdata$term,"relative_humidity"))])>=0,
  "+",
```

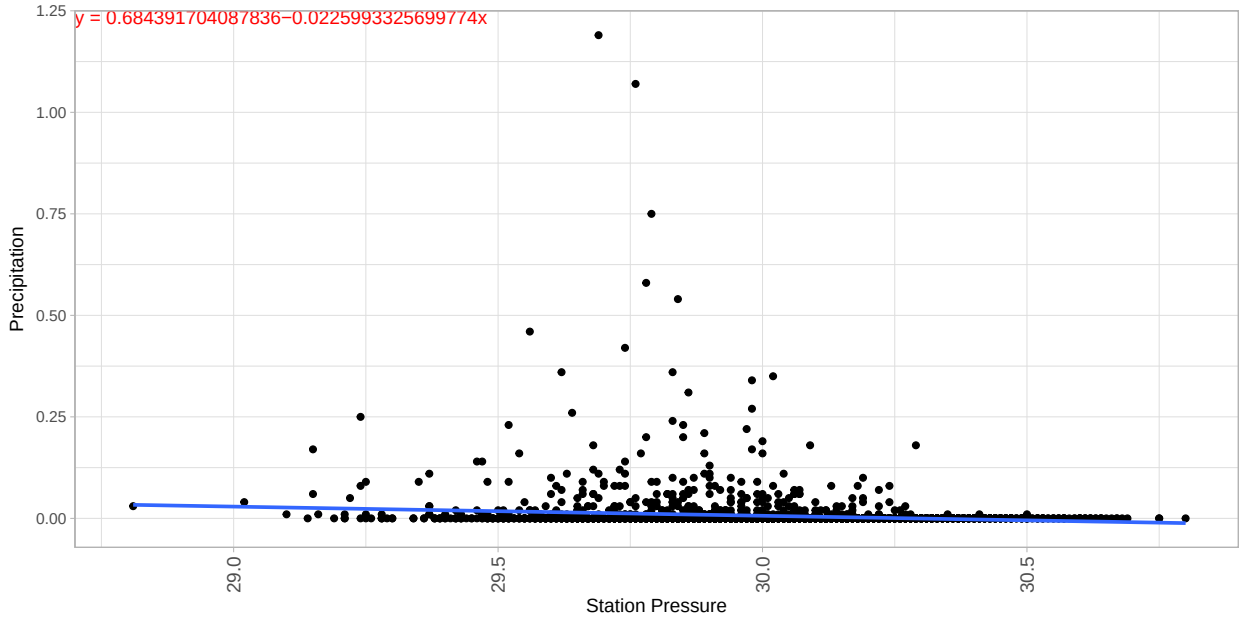
```
  "-")

reg_equation <- paste(
  "y = ",
  (fitdata$estimate[complete.cases(match(fitdata$term,"(Intercept)"))]),
  operator,
  abs(fitdata$estimate[complete.cases(match(fitdata$term,"relative_humidity"))]),
  "x",
  sep="")

p <- ggplot(data_subset_training,(aes(relative_humidity,precip))) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  xlab("Relative Humidity") +
  ylab(expression("Precipitation")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_blank(),
        legend.position = "none") +
annotate('text', label=reg_equation, x=-Inf, y=Inf, hjust=0, vjust=1,color="red")


p
```



Precipitation vs Station Pressure

```
lm_fit <- lm_model %>%
          fit(precip ~ station_pressure, data = data_subset_training)

fitdata <- tidy(lm_fit)
```

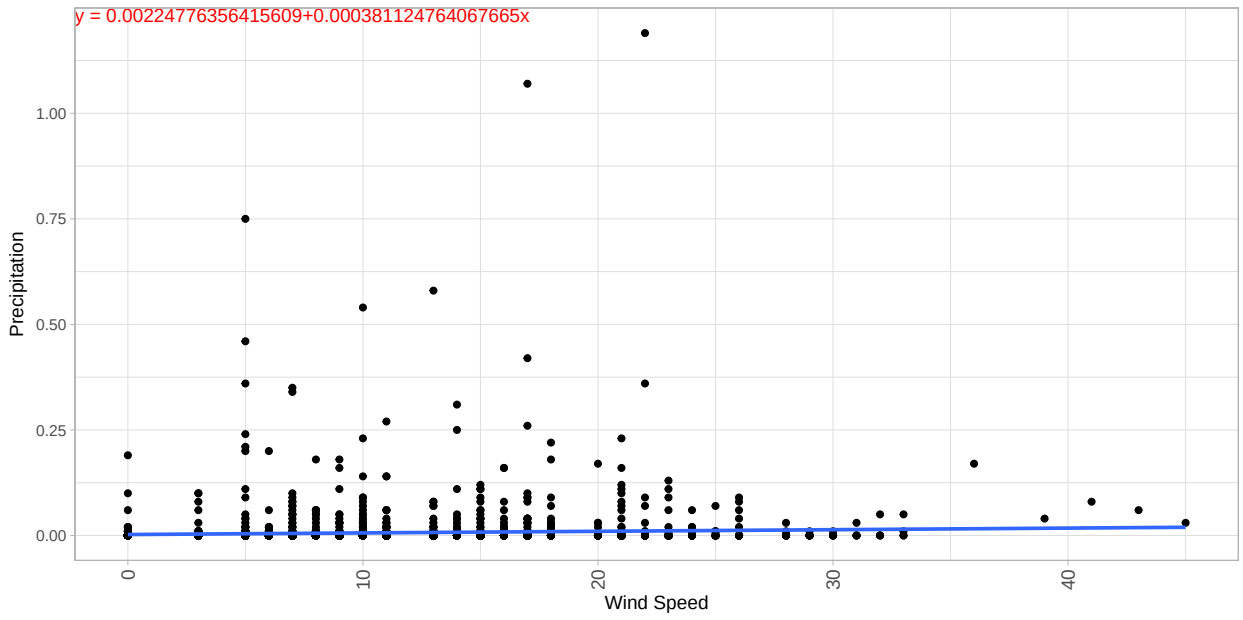
```
operator <- ifelse(as.numeric(fitdata$estimate[
  complete.cases(match(fitdata$term,"station_pressure"))])>=0,
  "+",
  "-")

reg_equation <- paste(
  "y = ",
  (fitdata$estimate[complete.cases(match(fitdata$term,"(Intercept)"))]),
  operator,
  abs(fitdata$estimate[complete.cases(match(fitdata$term,"station_pressure"))]),
  "x",
  sep="")

p <- ggplot(data_subset_training,(aes(station_pressure,precip))) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  xlab("Station Pressure") +
  ylab(expression("Precipitation")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_blank(),
        legend.position = "none") +
annotate('text', label=reg_equation, x=-Inf, y=Inf, hjust=0, vjust=1,color="red")


p
```



Precipitation vs Wind Speed

```
lm_fit <- lm_model %>%
          fit(precip ~ wind_speed, data = data_subset_training)
```

```
fitdata <- tidy(lm_fit)

operator <- ifelse(as.numeric(fitdata$estimate[
  complete.cases(match(fitdata$term,"wind_speed"))])>=0,
  "+",
  "-")

reg_equation <- paste(
  "y = ",
  (fitdata$estimate[complete.cases(match(fitdata$term,"(Intercept)"))]),
  operator,
  abs(fitdata$estimate[complete.cases(match(fitdata$term,"wind_speed"))]),
  "x",
  sep="")

p <- ggplot(data_subset_training,(aes(wind_speed,precip))) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  xlab("Wind Speed") +
  ylab(expression("Precipitation")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_blank(),
        legend.position = "none") +
annotate('text', label=reg_equation, x=-Inf, y=Inf, hjust=0, vjust=1,color="red")


p
```



9. Improve the Model

Now, try improving the simple models you created in the previous section.

Create at least two more models, each model should use at least one of the different techniques:

- Add more features/predictors
- Add regularization (L1, L2 or a mix)
- Add a polynomial component

Also, for each of the models you create, check the model performance using the training set and a metric like MSE, RMSE, or R-squared.

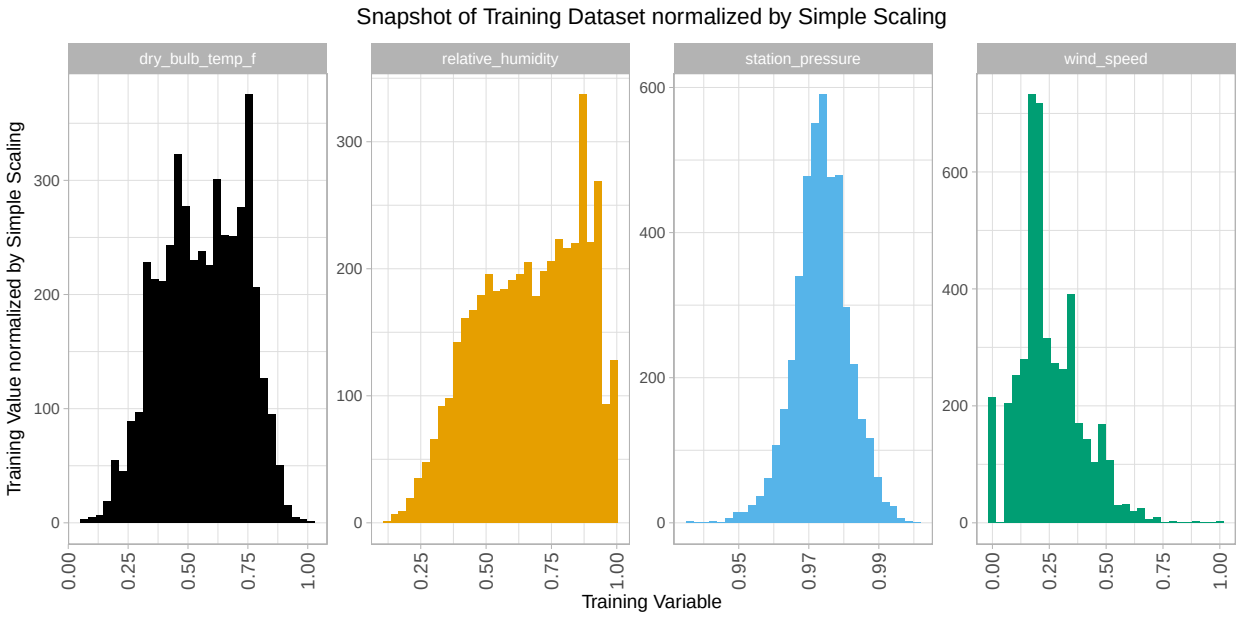
Consider using tidymodels if you choose to add regularization and tune lambda.

Normalize the data

Since the data are not normally distributed (See Section 7), we will need to normalize the data for a multiple regression model. We can use this recipe

Simple Scaling

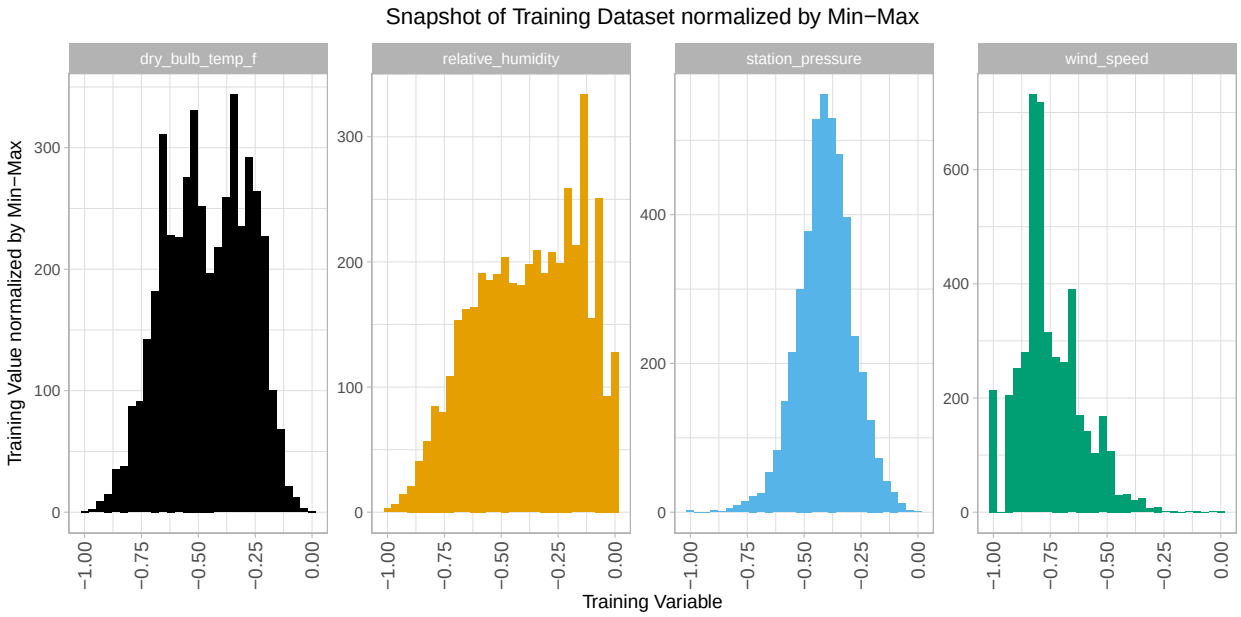
```
simple_scaled_data <- data_subset_training_plot %>%
  group_by(training_variable) %>%
  mutate(max_value = max(training_value,na.rm=TRUE),
         simple_scaled_value = training_value/max_value)

p <- ggplot(simple_scaled_data,aes(simple_scaled_value,fill=training_variable)) +
  geom_histogram() +
  facet_wrap(~training_variable,scales="free",ncol=5) +
  xlab("Training Variable") +
  ylab(expression("Training Value normalized by Simple Scaling")) +
  ggtitle(expression("Snapshot of Training Dataset normalized by Simple Scaling")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_text(size=rel(1.2), face="bold",hjust = 0.5),
        legend.position = "none")

p
```



These data are not normally distributed. The histograms should be centered on 0.50, but all histograms are skewed to some degree. Let's try a different method.

Min-Max

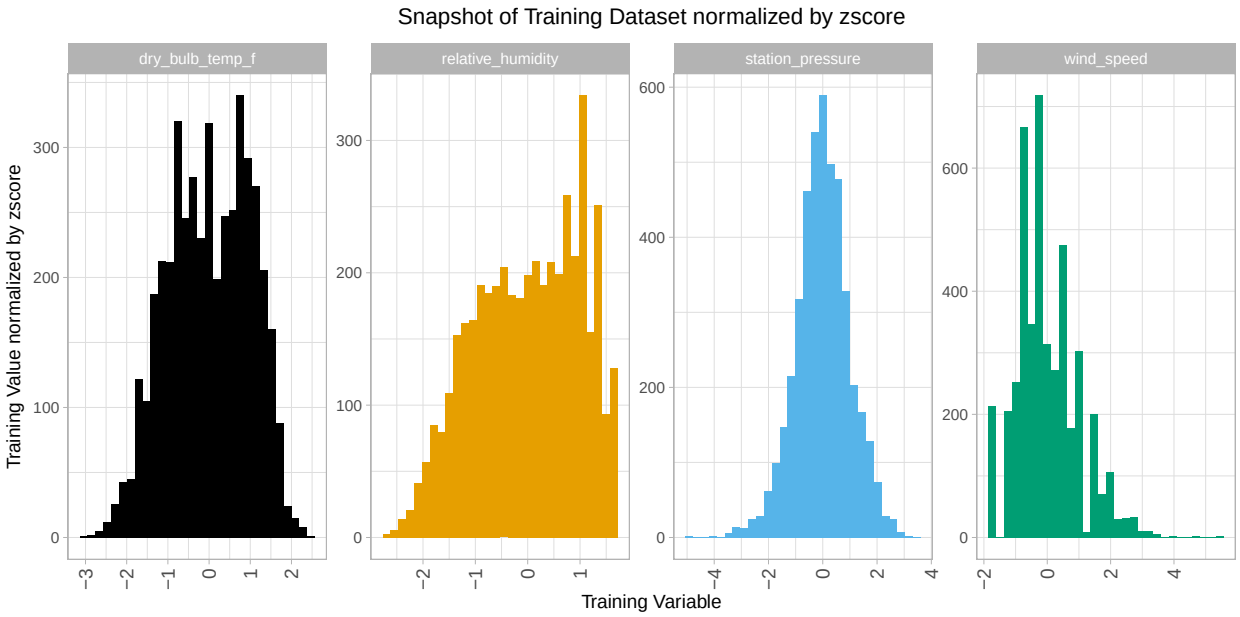
```
minmax_data <- data_subset_training_plot %>%
  group_by(training_variable) %>%
  mutate(max_value = max(training_value,na.rm=TRUE),
         min_value = min(training_value,na.rm=TRUE),
         minmax_value = (training_value-max_value)/(max_value-min_value))

p <- ggplot(minmax_data,aes(minmax_value,fill=training_variable)) +
  geom_histogram() +
  facet_wrap(~training_variable,scales="free",ncol=5) +
  xlab("Training Variable") +
  ylab(expression("Training Value normalized by Min-Max")) +
  ggtitle(expression("Snapshot of Training Dataset normalized by Min-Max")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_text(size=rel(1.2), face="bold",hjust = 0.5),
        legend.position = "none")

p
```



This looks a little better, but still has some skewness to it. Let's try one more method.

Z-Score

```
zscore_data <- data_subset_training_plot %>%
  group_by(training_variable) %>%
  mutate(mean_value = mean(training_value,na.rm=TRUE),
         sd_value = sd(training_value,na.rm=TRUE),
         zscore_value = (training_value-mean_value)/(sd_value))

p <- ggplot(zscore_data,aes(zscore_value,fill=training_variable)) +
  geom_histogram() +
  facet_wrap(~training_variable,scales="free",ncol=5) +
  xlab("Training Variable") +
  ylab(expression("Training Value normalized by zscore")) +
  ggtitle(expression("Snapshot of Training Dataset normalized by zscore")) +
  theme_light() +
  scale_color_colorblind() +
  scale_fill_colorblind() +
  theme(axis.text.x=element_text(size=rel(1.2),angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_text(size=rel(1.2), face="bold",hjust = 0.5),
        legend.position = "none")

p
```



This is the best normalization method, as all histograms are centered on 0, even if they are not perfectly normal.

Although I like this one, I'll incorporate all three into different models to see which method works the best.

Model Regulatization

Fit a new model

```
## Design Model

glm_model_L1 <- logistic_reg(
  mode = "classification",
  engine = "glm",
  penalty = NULL,
  mixture = 1
)

glm_model_L2 <- logistic_reg(
  mode = "classification",
  engine = "glm",
  penalty = NULL,
  mixture = 0
)

glm_model_50 <- logistic_reg(
  mode = "classification",
  engine = "glm",
  penalty = NULL,
  mixture = 0.5
)
```

Trying a different model method (Random Forest)