IBM-DS-with-R-2-Data-Analysis-with-R-NOAA

April 15, 2023

1 Assignment: Notebook for Peer Assignment

Estimated time needed: 60 minutes

2 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.

3 Introduction

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.

3.1 Table of Contents:

Using this R notebook you will complete **10 tasks**: * 0. Import Modules * 1. Download and Unzip NOAA Weather Dataset * 2. Read Dataset into Project * 3. Select Subset of Columns * 4. Clean Up Columns * 5. Convert Columns to Numerical Types * 6. Rename Columns * 7. Exploratory Data Analysis * 8. Linear Regression * 9. Improve the Model * 10. Find Best Model

0. Import required modules

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

[1]:
```
# Install tidymodels if you haven't done so
install.packages("rlang")
install.packages("tidymodels")
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
Warning message in install.packages("tidymodels"):
"installation of package 'tidymodels' had non-zero exit status"Updating HTML
index of packages in '.Library'
Making 'packages.html' … done
```

Note: After installing the packages, restart the kernel. Without installing the packages again, load them. Tidyverse and Tidymodels will be the two main packages you will use.

[2]:
```
# Library for modeling
library(tidymodels)

# Load tidyverse
library(tidyverse)
```

```
 Attaching packages                    tidymodels 1.0.0
broom        1.0.3      recipes      1.0.5
dials        1.1.0      rsample      1.1.1
dplyr        1.1.0      tibble       3.1.8
ggplot2      3.4.1      tidyr        1.3.0
infer        1.0.4      tune         1.0.1
modeldata    1.1.0      workflows    1.1.3
parsnip      1.0.4      workflowsets 1.0.0
purrr        1.0.1      yardstick    1.1.0
 Conflicts                       tidymodels_conflicts()
purrr::discard() masks scales::discard()
dplyr::filter()  masks stats::filter()
dplyr::lag()     masks stats::lag()
recipes::step()  masks stats::step()
• Learn how to get started at https://www.tidymodels.org/start/
 Attaching packages                     tidyverse 1.3.0
readr   1.3.1      forcats 0.5.0
stringr 1.5.0
 Conflicts                         tidyverse_conflicts()
readr::col_factor() masks scales::col_factor()
purrr::discard()    masks scales::discard()
dplyr::filter()     masks stats::filter()
stringr::fixed()    masks recipes::fixed()
dplyr::lag()        masks stats::lag()
readr::spec()       masks yardstick::spec()
```

3.1.1 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.

3.2 1. Download NOAA Weather Dataset

Use the `download.file()` function to download the sample dataset from the URL below.

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

[10]:
```
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, destfile = "noaa-weather-sample-data.tar.gz")
```

Untar the zipped file.

[12]:
```
untar('noaa-weather-sample-data.tar.gz', tar='internal')
```

```
Warning message in untar2(tarfile, files, list, exdir, restore_times):
"using pax extended headers"
```

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".

[13]:
```
df <- read.csv("noaa-weather-sample-data/jfk_weather_sample.csv")
```

Next, display the first few rows of the dataframe.

[14]:
```
head(df)
```

A data.frame: 6 × 9

	DATE <fct>	HOURLYDewPointTempF <fct>	HOURLYRelativeHumidity <int>	HOU <int
1	2015-07-25T13:51:00Z	60	46	83
2	2016-11-18T23:51:00Z	34	48	53
3	2013-01-06T08:51:00Z	33	89	36
4	2011-01-27T16:51:00Z	18	48	36
5	2015-01-03T12:16:00Z	27	61	39
6	2013-02-15T20:51:00Z	35	79	41

Also, take a `glimpse` of the dataset to see the different column data types and make sure it is the correct subset dataset with about 5700 rows and 9 columns.

[15]:
```
glimpse(df)
```

```
Rows: 5,727
Columns: 9
$ DATE                   <fct> 2015-07-25T13:51:00Z, 2016-11-18T23:51:00Z, 201…
$ HOURLYDewPointTempF    <fct> 60, 34, 33, 18, 27, 35, 4, 14, 51, 71, 76, 19, …
$ 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           <fct> 0.00, 0.00, 0.00, 0.00, T, 0.00, 0.00, 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.

[16]:
```
df <- df %>% select(HOURLYRelativeHumidity, HOURLYDRYBULBTEMPF, HOURLYPrecip,␣
↪HOURLYWindSpeed, HOURLYStationPressure)
```

Show the first 10 rows of this new dataframe.

[17]:
```
head(df,10)
```

A data.frame: 10 × 5

	HOURLYRelativeHumidity <int>	HOURLYDRYBULBTEMPF <int>	HOURLYPrecip <fct>	HOU <int>
1	46	83	0.00	13
2	48	53	0.00	6
3	89	36	0.00	13
4	48	36	0.00	14
5	61	39	T	11
6	79	41	0.00	6
7	51	19	0.00	0
8	65	24	0.00	11
9	90	54	0.06	11
10	94	73	NA	5

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.

[18]:
```
unique(df$HOURLYPrecip)
```

1. 0.00 2. T 3. 0.06 4. <NA> 5. 0.03 6. 0.02 7. 0.08 8. 0.01 9. 0.07 10. 0.16 11. 0.09 12. 0.22 13. 0.02s 14. 0.24 15. 0.18 16. 0.05 17. 0.04 18. 0.09s 19. 0.11 20. 0.14 21. 0.25 22. 0.10 23. 0.01s 24. 0.58 25. 0.12 26. 0.13 27. 0.46 28. 1.07 29. 1.19 30. 0.34 31. 0.20 32. 0.36s 33. 0.42 34. 0.17 35. 0.27 36. 0.35 37. 0.31 38. 0.33 39. 0.23 40. 0.26 41. 0.28 42. 0.75 43. 0.19 44. 0.36 45. 0.03s 46. 0.07s 47. 0.54 48. 0.59 49. 0.21

Levels: 1. '0.00' 2. '0.01' 3. '0.01s' 4. '0.02' 5. '0.02s' 6. '0.03' 7. '0.03s' 8. '0.04' 9. '0.05' 10. '0.06' 11. '0.07' 12. '0.07s' 13. '0.08' 14. '0.09' 15. '0.09s' 16. '0.10' 17. '0.11' 18. '0.12' 19. '0.13' 20. '0.14' 21. '0.16' 22. '0.17' 23. '0.18' 24. '0.19' 25. '0.20' 26. '0.21' 27. '0.22' 28. '0.23' 29. '0.24' 30. '0.25' 31. '0.26' 32. '0.27' 33. '0.28' 34. '0.31' 35. '0.33' 36. '0.34' 37. '0.35' 38. '0.36' 39. '0.36s' 40. '0.42' 41. '0.46' 42. '0.54' 43. '0.58' 44. '0.59' 45. '0.75' 46. '1.07' 47. '1.19' 48. 'T'

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 at 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.

[19]:
```
df <- df %>% mutate(HOURLYPrecip = str_remove(HOURLYPrecip,
                                              pattern = "s$"),
                    HOURLYPrecip = str_replace(HOURLYPrecip,
                                               "T", "0.0")) %>%
                     replace(is.na(.), 0)
unique(df$HOURLYPrecip)
```

1. '0.00' 2. '0.0' 3. '0.06' 4. '0' 5. '0.03' 6. '0.02' 7. '0.08' 8. '0.01' 9. '0.07' 10. '0.16' 11. '0.09' 12. '0.22' 13. '0.24' 14. '0.18' 15. '0.05' 16. '0.04' 17. '0.11' 18. '0.14' 19. '0.25' 20. '0.10' 21. '0.58' 22. '0.12' 23. '0.13' 24. '0.46' 25. '1.07' 26. '1.19' 27. '0.34' 28. '0.20' 29. '0.36' 30. '0.42' 31. '0.17' 32. '0.27' 33. '0.35' 34. '0.31' 35. '0.33' 36. '0.23' 37. '0.26' 38. '0.28' 39. '0.75' 40. '0.19' 41. '0.54' 42. '0.59' 43. '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.

[20]:
```
glimpse(df)
```

```
Rows: 5,727
Columns: 5
$ HOURLYRelativeHumidity <dbl> 46, 48, 89, 48, 61, 79, 51, 65, 90, 94, 79, 37,…
$ HOURLYDRYBULBTEMPF     <dbl> 83, 53, 36, 36, 39, 41, 19, 24, 54, 73, 83, 44,…
$ HOURLYPrecip           <chr> "0.00", "0.00", "0.00", "0.00", "0.0", "0.00", …
$ HOURLYWindSpeed        <dbl> 13, 6, 13, 14, 11, 6, 0, 11, 11, 5, 21, 7, 17, …
$ HOURLYStationPressure  <dbl> 29.99, 30.03, 30.12, 29.80, 30.50, 29.92, 30.40…
```

Convert `HOURLYPrecip` to the `numeric` type and store the cleaned dataframe as a new variable.

[21]:
```
df$HOURLYPrecip <- as.numeric(df$HOURLYPrecip)
```

We can now see that all fields have numerical data type.

[22]:
```
glimpse(df)
```

```
Rows: 5,727
Columns: 5
$ HOURLYRelativeHumidity <dbl> 46, 48, 89, 48, 61, 79, 51, 65, 90, 94, 79, 37,…
$ HOURLYDRYBULBTEMPF     <dbl> 83, 53, 36, 36, 39, 41, 19, 24, 54, 73, 83, 44,…
$ HOURLYPrecip           <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,…
$ HOURLYWindSpeed        <dbl> 13, 6, 13, 14, 11, 6, 0, 11, 11, 5, 21, 7, 17, …
$ HOURLYStationPressure  <dbl> 29.99, 30.03, 30.12, 29.80, 30.50, 29.92, 30.40…
```

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.

[23]:
```
df <- df %>% rename(relative_humidity=HOURLYRelativeHumidity,
                    dry_bulb_temp_f=HOURLYDRYBULBTEMPF,
                    precip=HOURLYPrecip,
                    wind_speed=HOURLYWindSpeed,
                    station_pressure=HOURLYStationPressure)
```

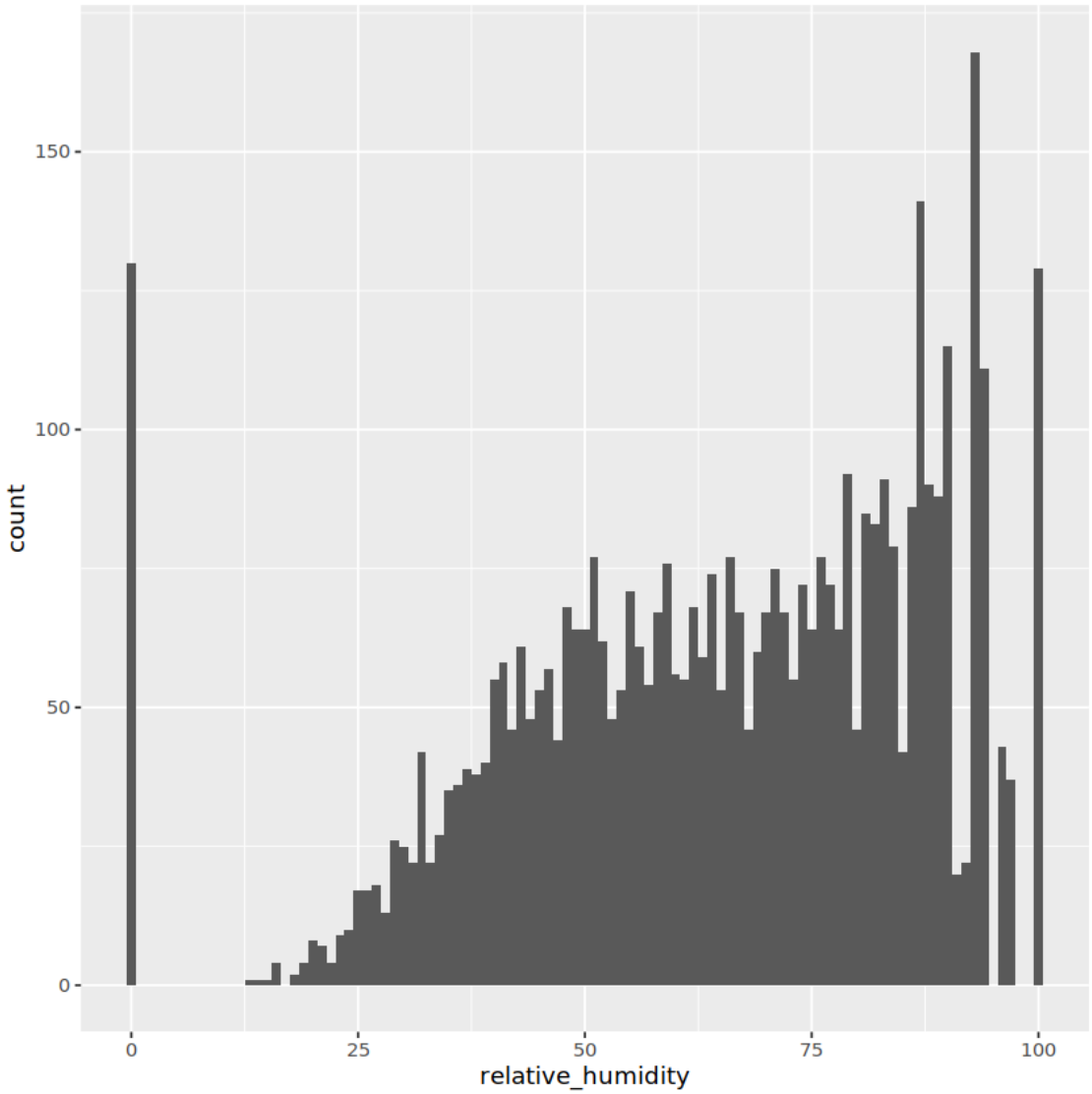
7. Exploratory Data Analysis Now that you have finished preprocessing the dataset, you can can start exploring the columns more.

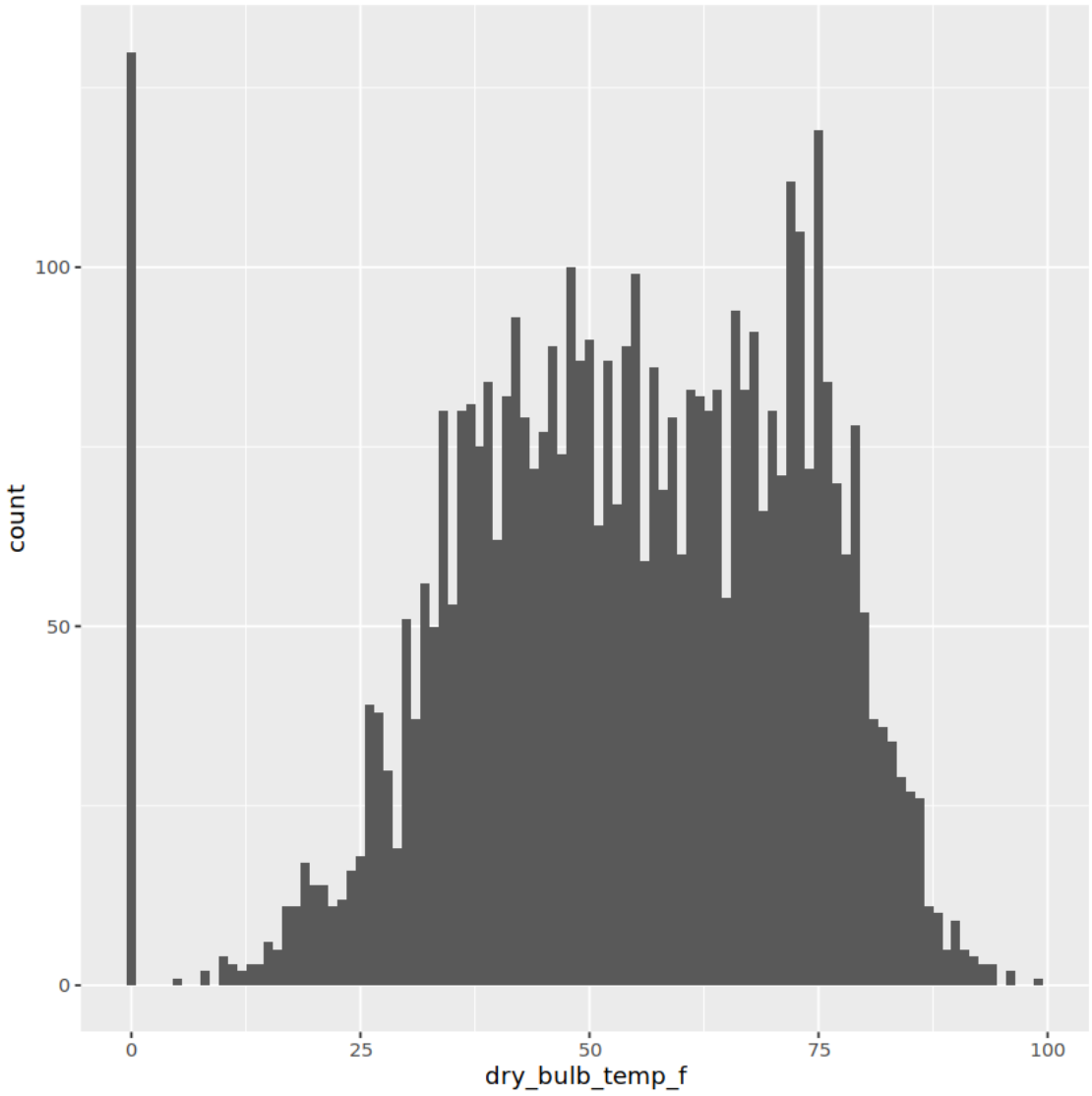
First, split the data into a training and testing set. Splitting a dataset is done randomly, so to have reproducible results set the seed = 1234. Also, use 80% of the data for training.

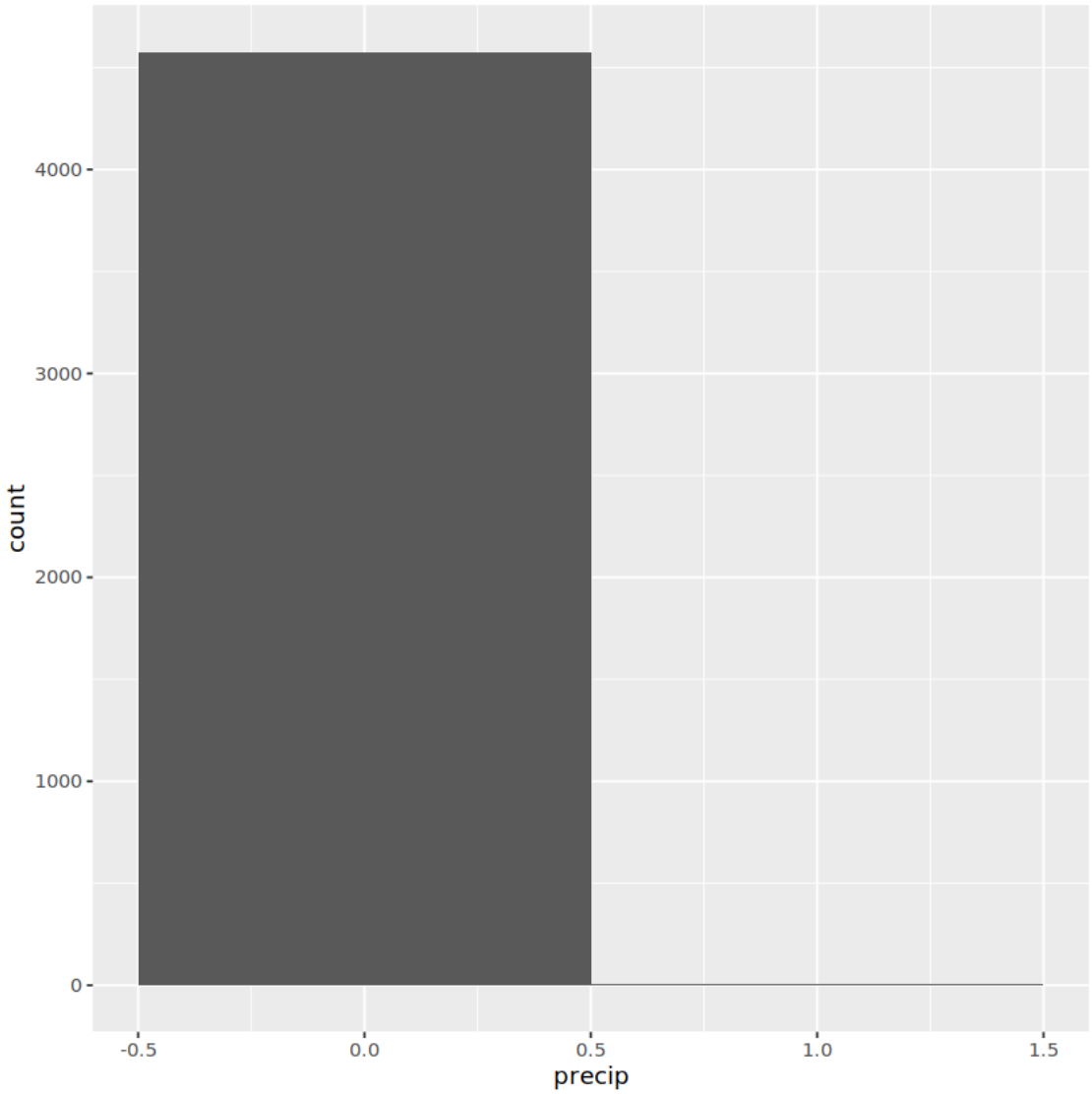
[24]:
```
set.seed(1234)
df_split <- initial_split(df, prop = 0.8)
train_data <- training(df_split)
test_data <- testing(df_split)
```

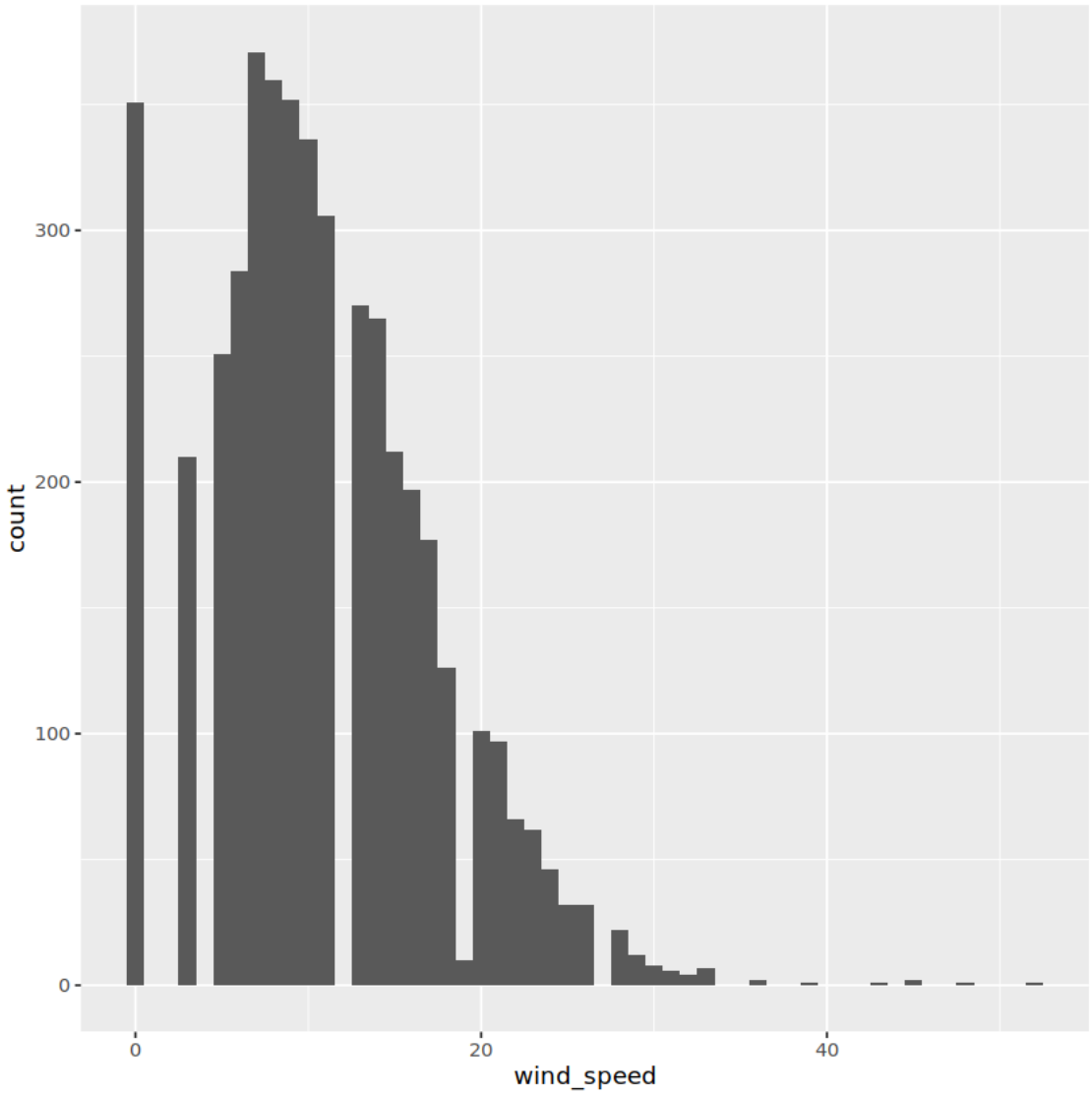
Next, looking at just the **training set**, plot histograms or box plots of the variables (`relative_humidity`, `dry_bulb_temp_f`, `precip`, `wind_speed`, `station_pressure`) for an intial look of their distributions using `tidyverse`'s `ggplot`. Leave the testing set as is because it is good practice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

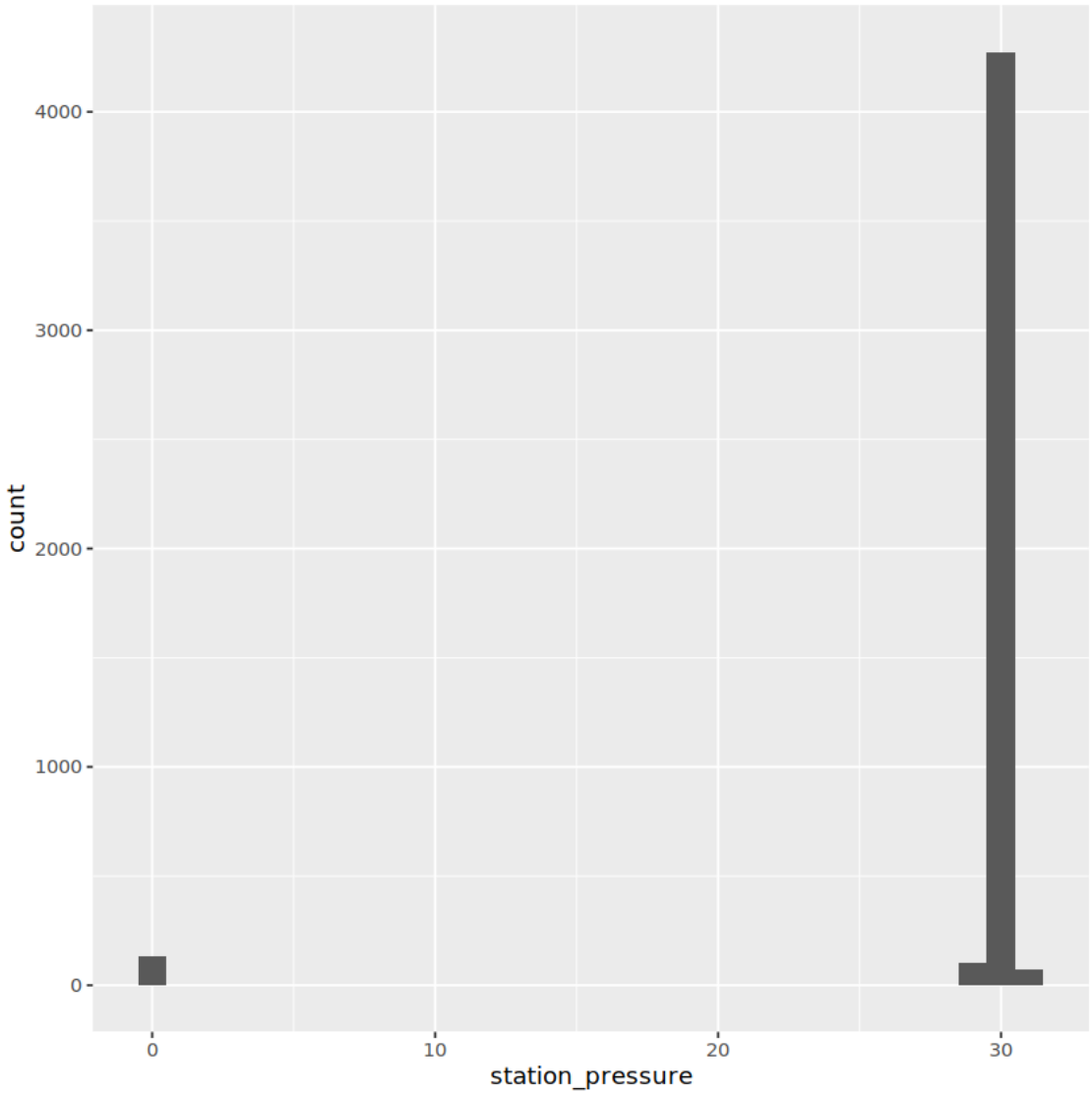
[25]:
```
train_data %>% ggplot(aes(relative_humidity))+geom_histogram(binwidth = 1)
train_data %>% ggplot(aes(dry_bulb_temp_f))+geom_histogram(binwidth = 1)
train_data %>% ggplot(aes(precip))+geom_histogram(binwidth = 1)
train_data %>% ggplot(aes(wind_speed))+geom_histogram(binwidth = 1)
train_data %>% ggplot(aes(station_pressure))+geom_histogram(binwidth = 1)
```


150
100
50
0
count
0
25
50
75
100
relative_humidity


count
100
50
0
0
25
50
75
100
dry_bulb_temp_f


4000
3000
2000
1000
0
count
-0.5
0.0
0.5
1.0
1.5
precip


count
300
200
100
0
0
20
40
wind_speed



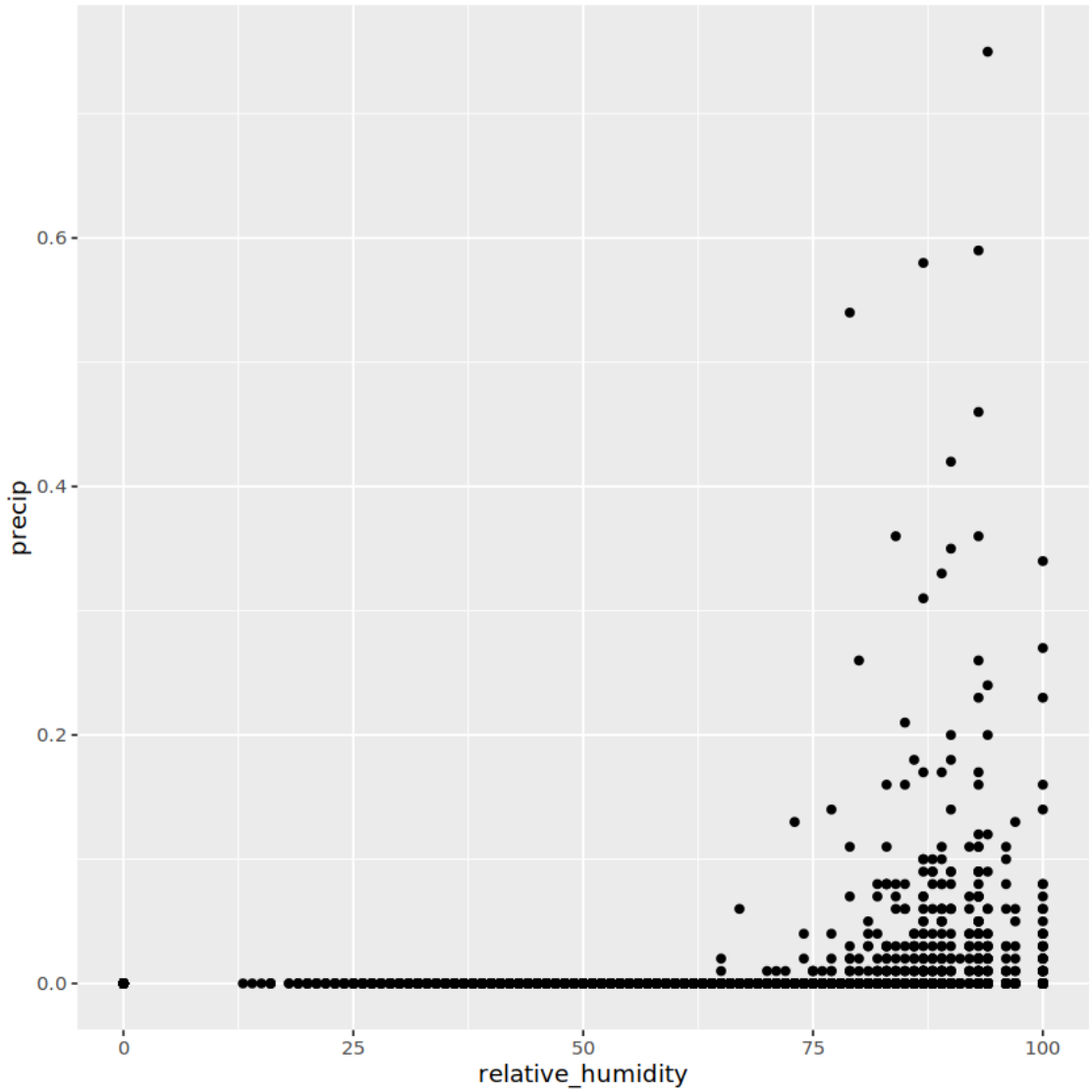
8. Linear Regression After exploring the dataset more, you are now ready to start creating models to predict the precipitation (`precip`).

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.

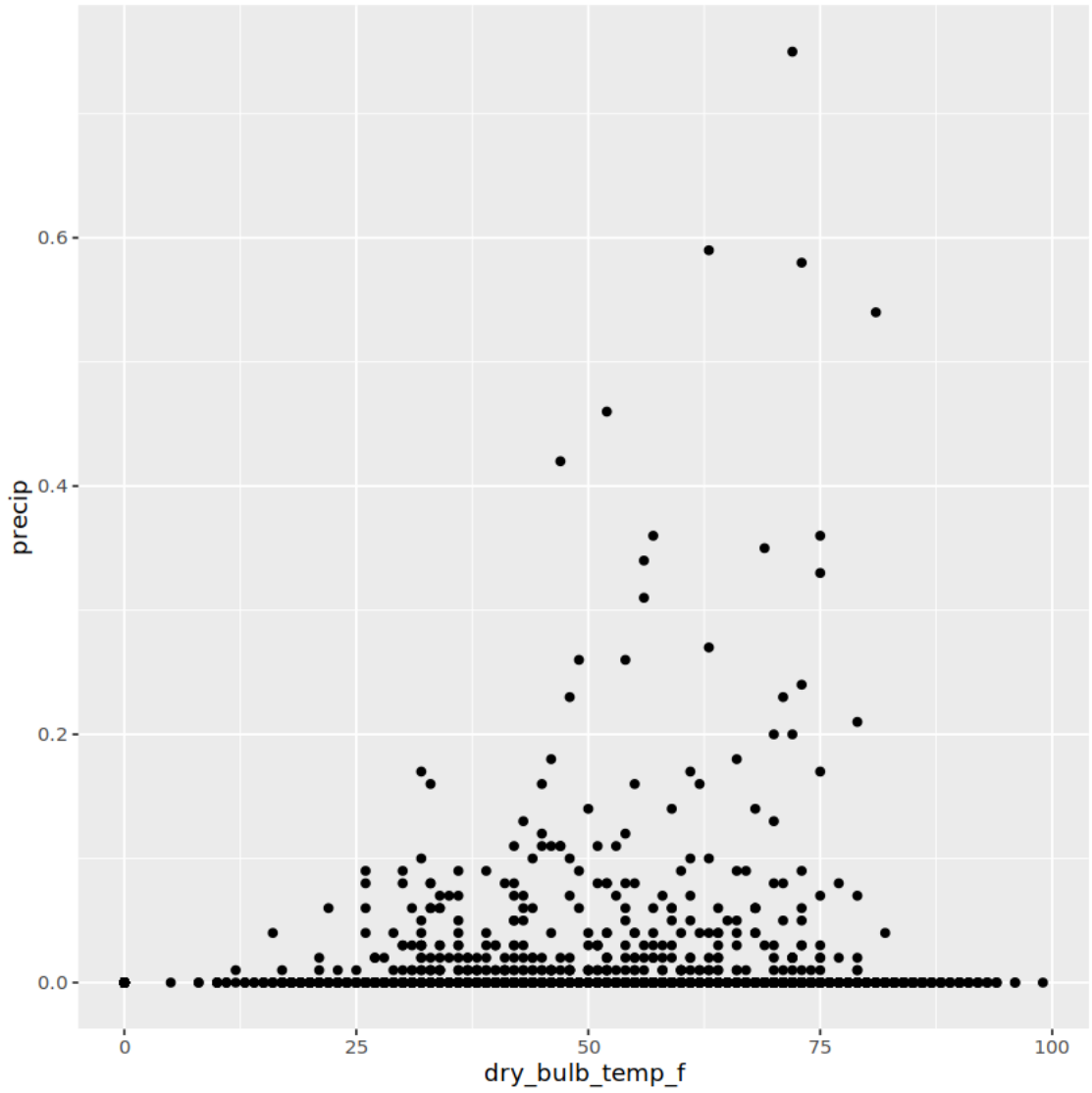
[26]:
```
model_humid <- lm(precip ~ relative_humidity, data = train_data)
summary(model_humid)
train_data %>% ggplot(aes(x=relative_humidity, y=precip))+geom_point()
```

```
Call:
```

```
lm(formula = precip ~ relative_humidity, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.01135 -0.00748 -0.00382  0.00005  0.73987

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       -9.007e-03  1.301e-03  -6.921  5.1e-12 ***
relative_humidity  2.035e-04  1.889e-05  10.773  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02904 on 4579 degrees of freedom
Multiple R-squared:  0.02472,        Adjusted R-squared:  0.02451
F-statistic: 116.1 on 1 and 4579 DF,  p-value: < 2.2e-16
```



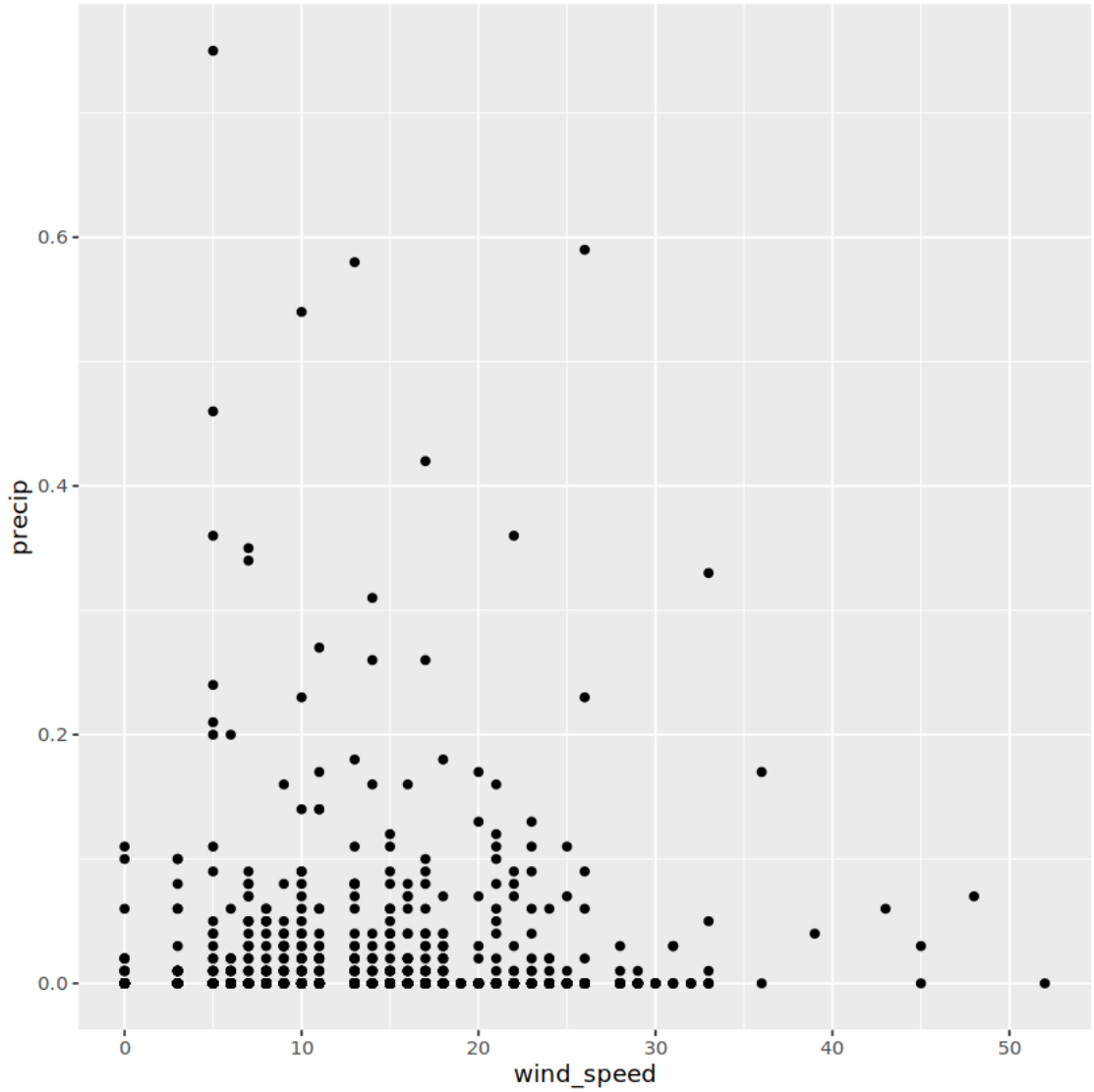
[27]:
```
model_dry <- lm(precip ~ dry_bulb_temp_f, data = train_data)
train_data %>% ggplot(aes(x=dry_bulb_temp_f, y=precip))+geom_point()
```



[28]:
```
model_wind <- lm(precip ~ wind_speed, data = train_data)
summary(model_wind)
train_data %>% ggplot(aes(x=wind_speed, y=precip))+geom_point()
```

```
Call:
lm(formula = precip ~ wind_speed, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.01583 -0.00511 -0.00370 -0.00286  0.74742

Coefficients:
```

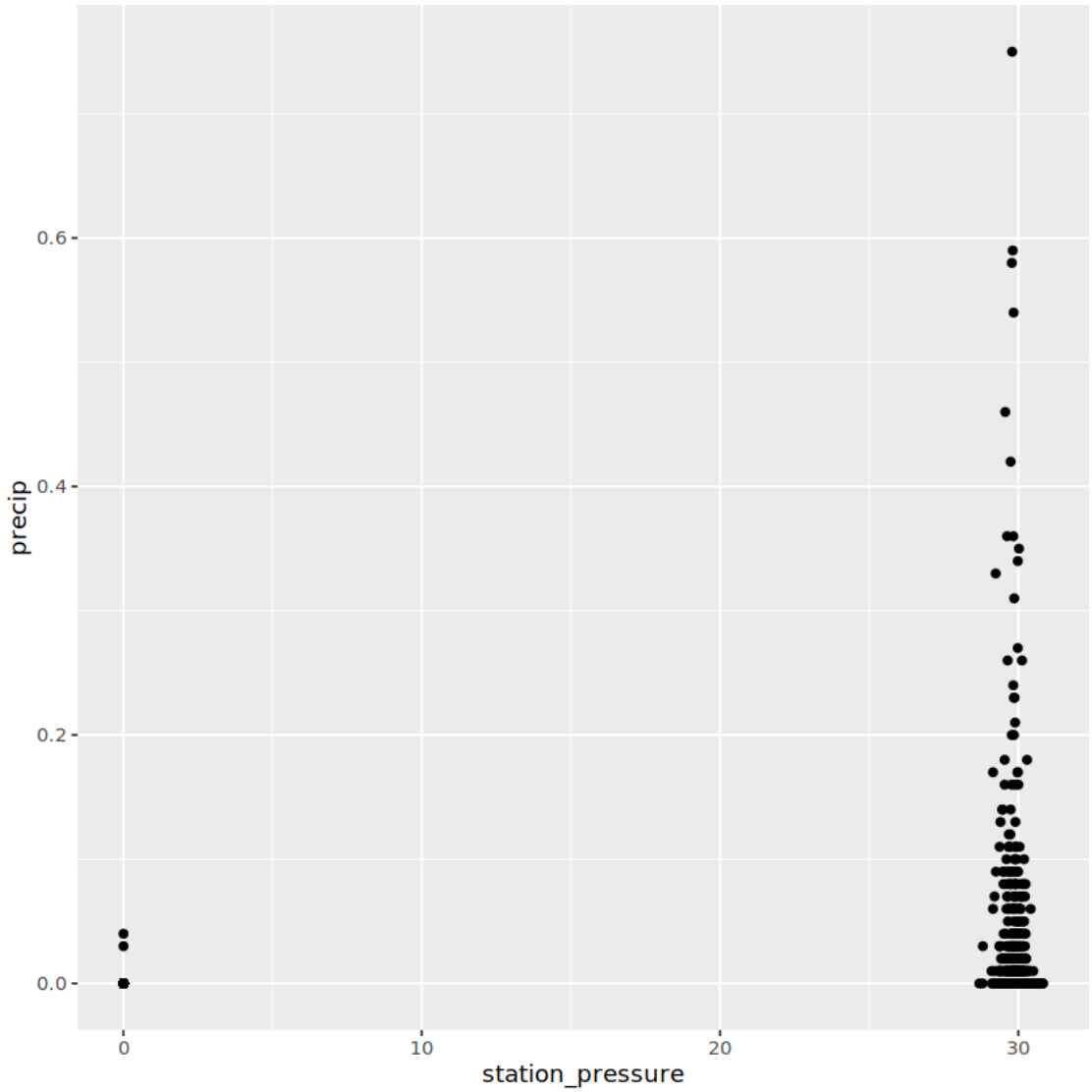
```
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.168e-03  8.452e-04   1.381    0.167
wind_speed  2.819e-04  6.681e-05   4.219  2.5e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02935 on 4579 degrees of freedom
Multiple R-squared:  0.003872,        Adjusted R-squared:  0.003654
F-statistic:  17.8 on 1 and 4579 DF,  p-value: 2.504e-05
```



[29]:
```
model_pressure <- lm(precip ~ station_pressure, data = train_data)
summary(model_pressure)
```

```
train_data %>% ggplot(aes(x=station_pressure, y=precip))+geom_point()
```

```
Call:
lm(formula = precip ~ station_pressure, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.00440 -0.00433 -0.00431 -0.00430  0.74571

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.383e-03  2.528e-03   0.547    0.584
station_pressure 9.775e-05  8.556e-05   1.143    0.253

Residual standard error: 0.02941 on 4579 degrees of freedom
Multiple R-squared:  0.000285,        Adjusted R-squared:  6.666e-05
F-statistic: 1.305 on 1 and 4579 DF,  p-value: 0.2533
```



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: 1. Add more features/predictors 2. Add regularization (L1, L2 or a mix) 3. 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.

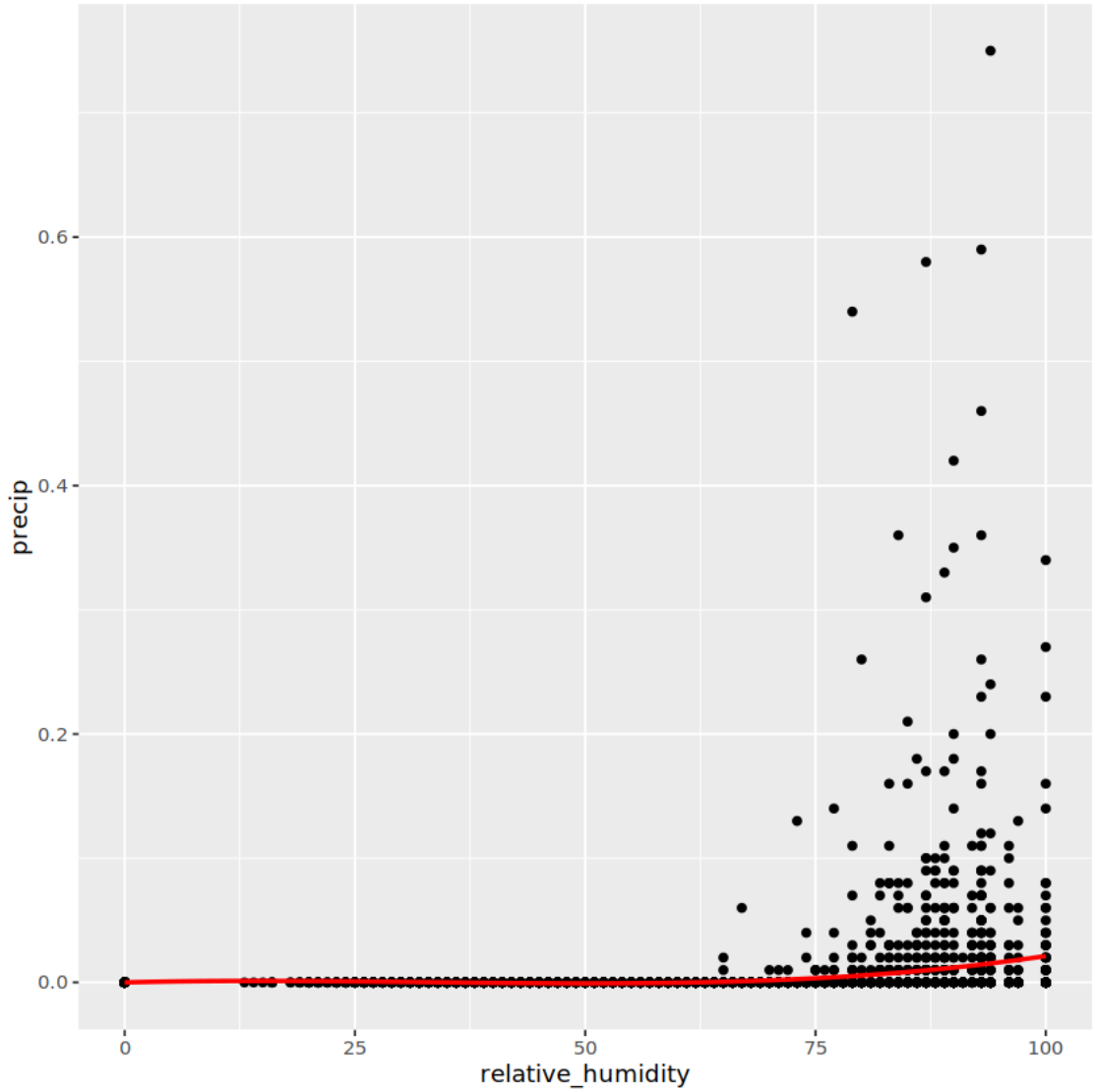
[30]:
```
mlr <- lm(precip~relative_humidity+wind_speed+station_pressure,
          data = train_data)
summary(mlr)
```

```
Call:
lm(formula = precip ~ relative_humidity + wind_speed + station_pressure,
    data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.03909 -0.00725 -0.00322  0.00077  0.74092

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       -6.763e-04  2.485e-03  -0.272    0.786
relative_humidity  2.779e-04  2.167e-05  12.823  < 2e-16 ***
wind_speed         4.735e-04  6.951e-05   6.813 1.08e-11 ***
station_pressure  -6.289e-04  9.992e-05  -6.294 3.39e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02885 on 4577 degrees of freedom
Multiple R-squared:  0.03842,        Adjusted R-squared:  0.03779
F-statistic: 60.95 on 3 and 4577 DF,  p-value: < 2.2e-16
```

[31]:
```
poly_reg <- lm(precip~poly(relative_humidity, 3),data = train_data)
summary(poly_reg)
ggplot(train_data, aes(x = relative_humidity, y = precip)) +
    geom_point() +
    geom_smooth(method = "lm",
                formula = y ~ poly(x, 3),
                col = "red", se = FALSE)
```

```
Call:
lm(formula = precip ~ poly(relative_humidity, 3), data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.02127 -0.00612 -0.00054  0.00046  0.73471

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 0.0042283  0.0004259   9.929  < 2e-16 ***
poly(relative_humidity, 3)1 0.3128793  0.0288235  10.855  < 2e-16 ***
poly(relative_humidity, 3)2 0.2195959  0.0288235   7.619  3.1e-14 ***
poly(relative_humidity, 3)3 0.1083739  0.0288235   3.760 0.000172 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.02882 on 4577 degrees of freedom
Multiple R-squared:  0.03986,        Adjusted R-squared:  0.03923
F-statistic: 63.34 on 3 and 4577 DF,  p-value: < 2.2e-16
```



[32]:
```
library(glmnet)
df_recipe <- recipe(precip ~ ., data = train_data)
ridge_spec <- linear_reg(penalty = 0.1, mixture = 0) %>% set_engine("glmnet")

ridge_wf <- workflow() %>% add_recipe(df_recipe)

ridge_fit <- ridge_wf %>% add_model(ridge_spec) %>% fit(data = train_data)
```

```
ridge_fit %>% pull_workflow_fit() %>% tidy()
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loading required package: foreach

Attaching package: 'foreach'

The following objects are masked from 'package:purrr':

    accumulate, when

Loaded glmnet 2.0-18

Warning message:
"`pull_workflow_fit()` was deprecated in workflows 0.2.3.
 Please use `extract_fit_parsnip()` instead."
```

A tibble: 5 × 3

term <chr>	estimate <dbl>	penalty <dbl>
(Intercept)	5.146507e-04	0.1
relative_humidity	4.658964e-05	0.1
dry_bulb_temp_f	2.157180e-06	0.1
wind_speed	6.690713e-05	0.1
station_pressure	-5.443800e-06	0.1

10. Find Best Model Compare the regression metrics of each model from section 9 to find the best model overall. To do this,

1. Evaluate the models on the **testing set** using at least one metric (like MSE, RMSE or R-squared).
2. After calculating the metrics on the testing set for each model, print them out in as a table to easily compare. You can use something like:

```
model_names <- c("model_1", "model_2", "model_3")
train_error <- c("model_1_value", "model_2_value", "model_3_value")
test_error <- c("model_1_value", "model_2_value", "model_3_value")
comparison_df <- data.frame(model_names, train_error, test_error)
```

3. Finally, from the comparison table you create, conclude which model performed the best.

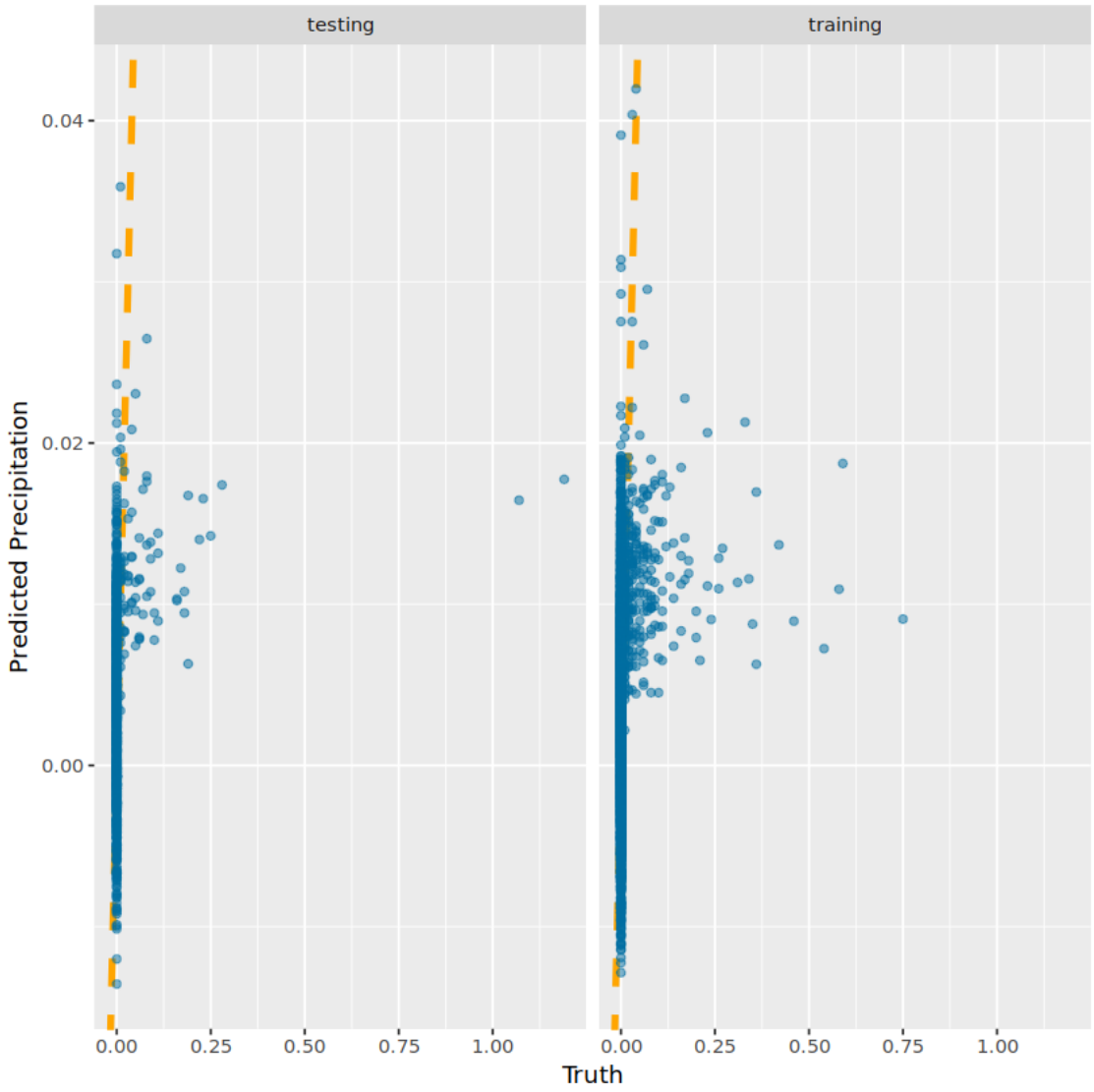
[33]:
```
train_fit_lm <- linear_reg() %>% set_engine("lm") %>%
  fit(precip~relative_humidity+wind_speed+station_pressure,
      data = train_data)
```

```
train_fit_poly <- linear_reg() %>% set_engine("lm") %>%
  fit(precip ~ poly(relative_humidity, 3),
      data = train_data)

train_fit_ridge <- linear_reg(penalty = 0.1, mixture = 0) %>%␣
 ↪set_engine("glmnet") %>%
  fit(precip~relative_humidity+wind_speed+station_pressure,
      data = train_data)
```

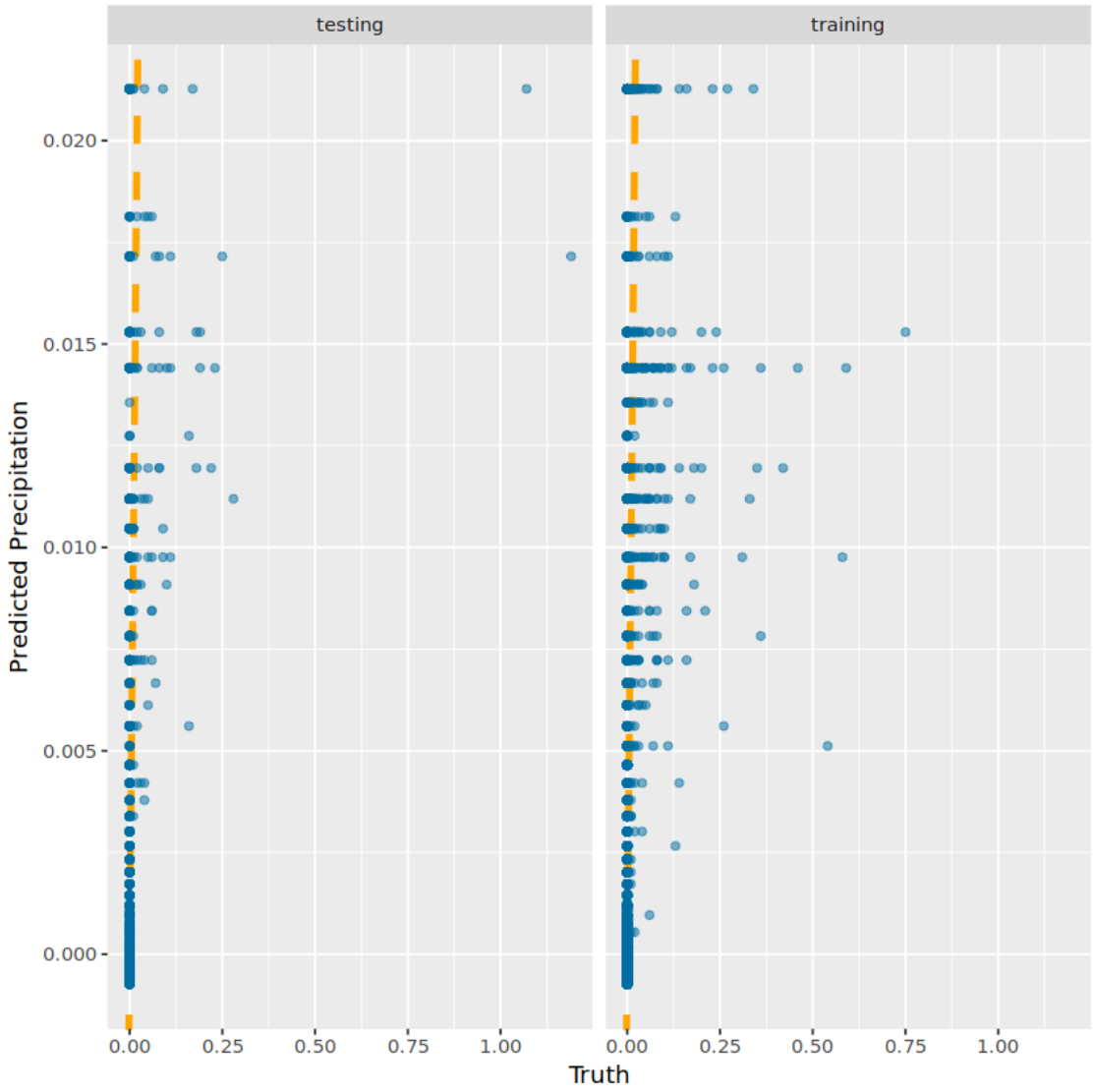
[35]:
```
train_results_lm <- train_fit_lm %>% predict(new_data = train_data) %>%
  mutate(truth = train_data$precip)

test_results_lm <- train_fit_lm %>% predict(new_data = test_data) %>%
  mutate(truth = test_data$precip)

train_results_poly <- train_fit_poly %>% predict(new_data = train_data) %>%
  mutate(truth = train_data$precip)

test_results_poly <- train_fit_poly %>% predict(new_data = test_data) %>%
  mutate(truth = test_data$precip)

train_results_ridge <- train_fit_ridge %>% predict(new_data = train_data) %>%
  mutate(truth = train_data$precip)

test_results_ridge <- train_fit_ridge %>% predict(new_data = test_data) %>%
  mutate(truth = test_data$precip)
```

[36]:
```
test_results_lm %>%
  mutate(train = "testing") %>%
  bind_rows(train_results_lm %>% mutate(train = "training")) %>%
  ggplot(aes(truth, .pred)) +
  geom_abline(lty = 2, color = "orange",
              size = 1.5) +
  geom_point(color = '#006EA1',
             alpha = 0.5) +
  facet_wrap(~train) +
  labs(x = "Truth",
       y = "Predicted Precipitation")
```

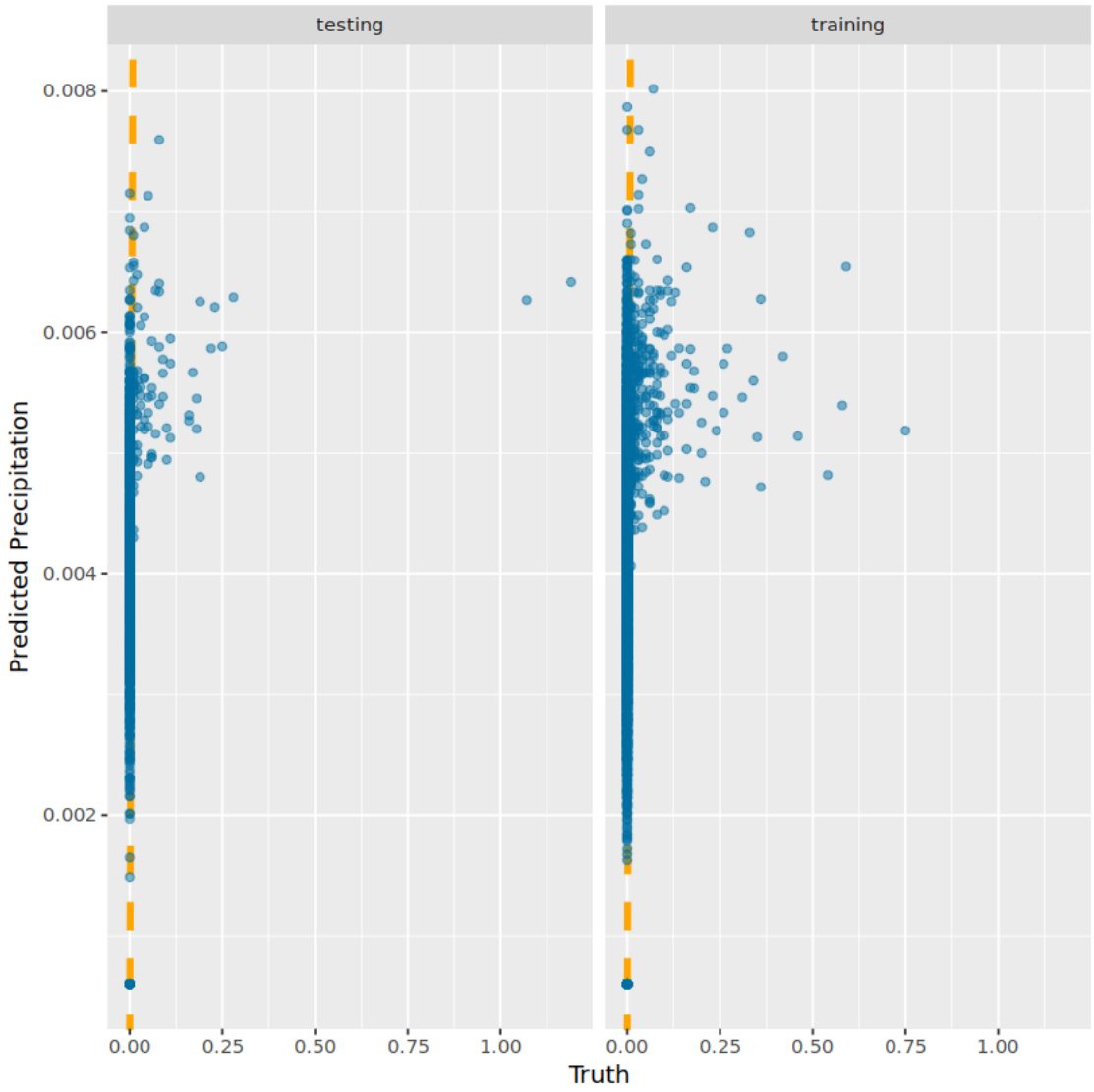
```
Warning message:
"Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
 Please use `linewidth` instead."
```



[37]:
```
test_results_poly %>%
  mutate(train = "testing") %>%
  bind_rows(train_results_poly %>% mutate(train = "training")) %>%
  ggplot(aes(truth, .pred)) +
  geom_abline(lty = 2, color = "orange",
              size = 1.5) +
  geom_point(color = '#006EA1',
             alpha = 0.5) +
  facet_wrap(~train) +
  labs(x = "Truth",
       y = "Predicted Precipitation")
```



[38]:
```
test_results_ridge %>%
  mutate(train = "testing") %>%
  bind_rows(train_results_ridge %>% mutate(train = "training")) %>%
  ggplot(aes(truth, .pred)) +
  geom_abline(lty = 2, color = "orange",
              size = 1.5) +
  geom_point(color = '#006EA1',
             alpha = 0.5) +
  facet_wrap(~train) +
  labs(x = "Truth",
       y = "Predicted Precipitation")
```



[39]:
```
train_error_lm <- rmse(train_results_lm, truth = truth,
     estimate = .pred)

test_error_lm <- rmse(test_results_lm, truth = truth,
     estimate = .pred)
```

[40]:
```
train_error_poly <- rmse(train_results_poly, truth = truth,
     estimate = .pred)

test_error_poly <- rmse(test_results_poly, truth = truth,
     estimate = .pred)
```

[41]:
```
train_error_ridge <- rmse(train_results_ridge, truth = truth,
     estimate = .pred)

test_error_ridge <- rmse(test_results_ridge, truth = truth,
     estimate = .pred)
```

[42]:
```
train_rsq_lm <- rsq(train_results_lm, truth = truth,
    estimate = .pred)

test_rsq_lm <- rsq(test_results_lm, truth = truth,
    estimate = .pred)
```

[43]:
```
train_rsq_poly <- rsq(train_results_poly, truth = truth,
    estimate = .pred)

test_rsq_poly <- rsq(test_results_poly, truth = truth,
    estimate = .pred)
```

[44]:
```
train_rsq_ridge <- rsq(train_results_ridge, truth = truth,
    estimate = .pred)

test_rsq_ridge <- rsq(test_results_ridge, truth = truth,
    estimate = .pred)
```

[45]:
```
model_names <- c("mlr", "poly", "ridge_L2")
train_error <- c(train_error_lm$.estimate, train_error_poly$.estimate,␣
 ↪train_error_ridge$.estimate)
test_error <- c(test_error_lm$.estimate, test_error_poly$.estimate,␣
 ↪test_error_ridge$.estimate)
train_rsq <- c(train_rsq_lm$.estimate, train_rsq_poly$.estimate,␣
 ↪train_rsq_ridge$.estimate)
test_rsq <- c(test_rsq_lm$.estimate, test_rsq_poly$.estimate, test_rsq_ridge$.
 ↪estimate)
comparison_df <- data.frame(model_names, train_error, test_error, train_rsq,␣
 ↪test_rsq)
comparison_df
```

A data.frame: 3 × 5

model_names <fct>	train_error <dbl>	test_error <dbl>	train_rsq <dbl>	test_rsq <dbl>
mlr	0.02883253	0.05160782	0.03841651	0.04198213
poly	0.02881088	0.05149772	0.03985982	0.05150889
ridge_L2	0.02922973	0.05229692	0.03060467	0.03498883

3.3 Author(s)

Yiwen Li

3.4 Contributions

Tiffany Zhu

##

[ ]: