Assignment: Notebook for Peer Assignment

Estimated time needed: 60 minutes

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.

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 (https://developer.ibm.com/exchanges/data/all/jfk-weather-data/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDA0151ENSkillsNetwork926-2022-01-01)](https://developer.ibm.com/exchanges/data/all/jfk-weather-data/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDA0151ENSkillsNetwork926-2022-01-01).

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.

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".

In [76]:
```
# 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
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.

In [77]:
```
# Library for modeling
library(tidymodels)

# Load tidyverse
library(tidyverse)
```

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

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. (https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weather-data-jfk-airport/1.1.4/noaa_weather.html)

For more information about the dataset, checkout the preview (https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weather-data-jfk-airport/1.1.4/data-preview/index.html?_ga=2.176781478.281508226.1616293518-1509963377.1616117067&cm_mc_uid=90945889189161532555498&cm_mc_sid_50200000=64650651616293516933) of NOAA Weather - JFK Airport

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

In [78]:
```
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.

In [79]:
```
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".

In [49]:
```
NOAAdf <-read.csv("noaa-weather-sample-data/jfk_weather_sample.csv")
```

Next, display the first few rows of the dataframe.

In [50]:
```
head(NOAAdf)
```

DATE	HOURLYDewPointTempF	HOURLYRelativeHumidity	HOURLYDRYBULBTEMPF	HOURLYWETBULBTEMPF	HOURLYPrecip	HOURLYWindSpeed	HOURLYSeaLevelPressure	HOURLYStationPressure
2015-07-25T13:51:00Z	60	46	83	68	0.00	13	30.01	29.99
2016-11-18T23:51:00Z	34	48	53	44	0.00	6	30.05	30.03
2013-01-06T08:51:00Z	33	89	36	35	0.00	13	30.14	30.12
2011-01-27T16:51:00Z	18	48	36	30	0.00	14	29.82	29.80
2015-01-03T12:16:00Z	27	61	39	34	T	11	NA	30.50
2013-02-15T20:51:00Z	35	79	41	38	0.00	6	29.94	29.92

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.

In [51]:
```
glimpse(NOAAdf)
```

```
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.

In [52]:
```
subsetNOAAdf <- select(NOAAdf, c("HOURLYRelativeHumidity",
"HOURLYDRYBULBTEMPF",
"HOURLYPrecip",
"HOURLYWindSpeed",
"HOURLYStationPressure"))
```

Show the first 10 rows of this new dataframe.

In [53]:
```
head(subsetNOAAdf,10)
```

HOURLYRelativeHumidity	HOURLYDRYBULBTEMPF	HOURLYPrecip	HOURLYWindSpeed	HOURLYStationPressure
46	83	0.00	13	29.99
48	53	0.00	6	30.03
89	36	0.00	13	30.12
48	36	0.00	14	29.80
61	39	T	11	30.50
79	41	0.00	6	29.92
51	19	0.00	0	30.40
65	24	0.00	11	30.35
90	54	0.06	11	30.03
94	73	NA	5	29.91

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.

In [54]:
```
unique(subsetNOAAdf$HOURLYPrecip)
```

0.00 T 0.06 <NA> 0.03 0.02 0.08 0.01 0.07 0.16 0.09 0.22 0.02s 0.24 0.18 0.05 0.04 0.09s 0.11 0.14 0.25 0.10 0.01s 0.58 0.12 0.13 0.46 1.07 1.19 0.34 0.20 0.36s 0.42 0.17 0.27 0.35 0.31 0.33 0.23 0.26 0.28 0.75 0.19 0.36 0.03s 0.07s 0.54 0.59 0.21

▶ **Levels**:

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 (https://www.rdocumentation.org/packages/stringi/versions/1.5.3/topics/about_search_regex?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDA0151ENSkillsNetwork926-2022-01-01) 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.

In [55]:
```
subset2NOAAdf <- subsetNOAAdf %>%
 mutate(HOURLYPrecip = str_replace_all(HOURLYPrecip, "T", "0.0"))

subset3NOAAdf <- subset2NOAAdf %>%
 mutate(HOURLYPrecip = str_remove(HOURLYPrecip, pattern = "s$"))

unique(subset3NOAAdf$HOURLYPrecip)
```

```
'0.00'  '0.0'  '0.06'  NA  '0.03'  '0.02'  '0.08'  '0.01'  '0.07'  '0.16'  '0.09'  '0.22'  '0.24'  '0.18'  '0.05'  '0.04'  '0.11'  '0.14'  '0.25'  '0.10'  '0.58'  '0.12'  '0.13'  '0.46'  '1.07'  '1.19'  '0.34'  '0.20'  '0.36'  '0.42'
'0.17'  '0.27'  '0.35'  '0.31'  '0.33'  '0.23'  '0.26'  '0.28'  '0.75'  '0.19'  '0.54'  '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.

In [56]:
```
glimpse(subset3NOAAdf)
```

```
Rows: 5,727
Columns: 5
$ 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,…
$ HOURLYPrecip           <chr> "0.00", "0.00", "0.00", "0.00", "0.0", "0.00", …
$ HOURLYWindSpeed        <int> 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.

In [57]:
```
subset3NOAAdf$HOURLYPrecip <- (as.numeric(subset3NOAAdf$HOURLYPrecip))
glimpse(subset3NOAAdf)
```

```
Rows: 5,727
Columns: 5
$ 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,…
$ HOURLYPrecip           <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,…
$ HOURLYWindSpeed        <int> 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…
```

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

In [58]:
```
glimpse(subset3NOAAdf)
```

```
Rows: 5,727
Columns: 5
$ 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,…
$ HOURLYPrecip           <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,…
$ HOURLYWindSpeed        <int> 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.

In [59]:
```
subset4NOAAdf <- subset3NOAAdf %>%
rename('relative_humidity' = 'HOURLYRelativeHumidity',
'dry_bulb_temp_f' = 'HOURLYDRYBULBTEMPF',
'precip' = 'HOURLYPrecip',
'wind_speed' = 'HOURLYWindSpeed',
'station_pressure' = 'HOURLYStationPressure')

glimpse(subset4NOAAdf)
```

```
Rows: 5,727
Columns: 5
$ relative_humidity <int> 46, 48, 89, 48, 61, 79, 51, 65, 90, 94, 79, 37, 72, …
$ dry_bulb_temp_f   <int> 83, 53, 36, 36, 39, 41, 19, 24, 54, 73, 83, 44, 57, …
$ precip            <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.06…
$ wind_speed        <int> 13, 6, 13, 14, 11, 6, 0, 11, 11, 5, 21, 7, 17, 8, 3,…
$ station_pressure  <dbl> 29.99, 30.03, 30.12, 29.80, 30.50, 29.92, 30.40, 30.…
```

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.

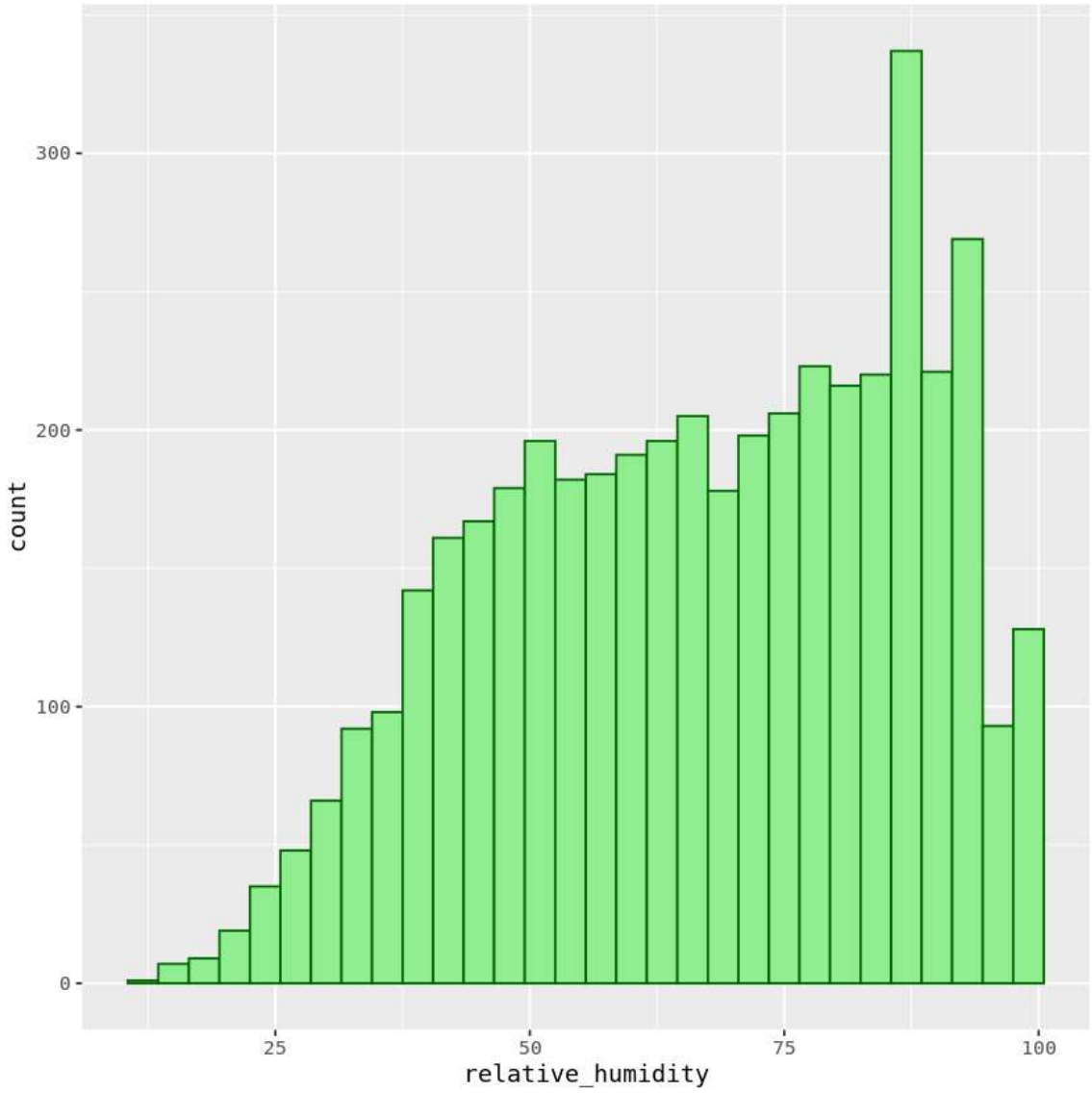
In [60]:
```
set.seed(1234)
train_NOAA <- initial_split(subset4NOAAdf, prop = 0.8)
train_set <- training(train_NOAA)
test_set <- testing(train_NOAA)
```

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.

In [61]:

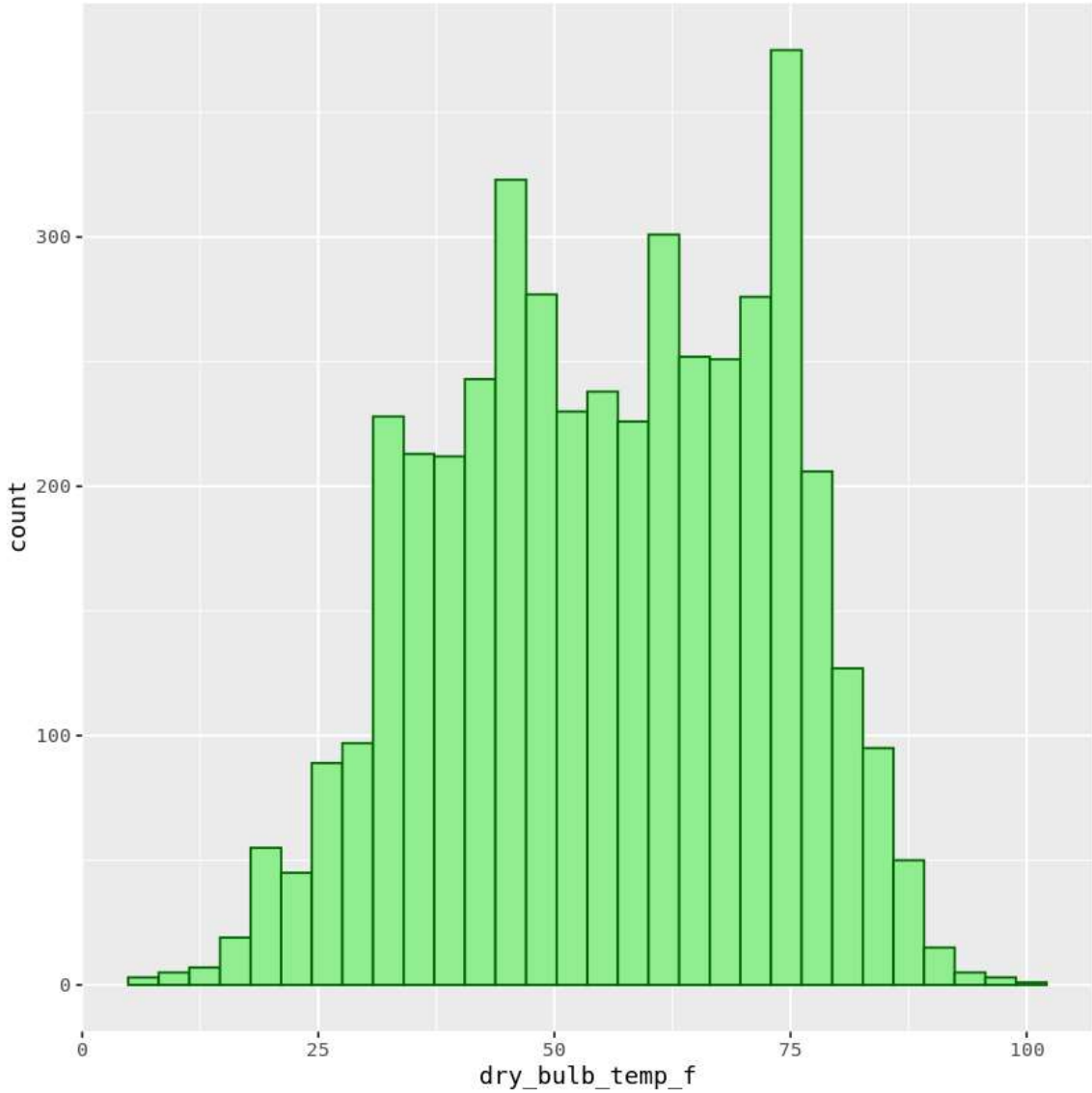
```
ggplot(train_set, aes(x = relative_humidity))+
geom_histogram(color = " darkgreen", fill = "lightgreen")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.Warning message:
“Removed 114 rows containing non-finite values (`stat_bin()`).”
```



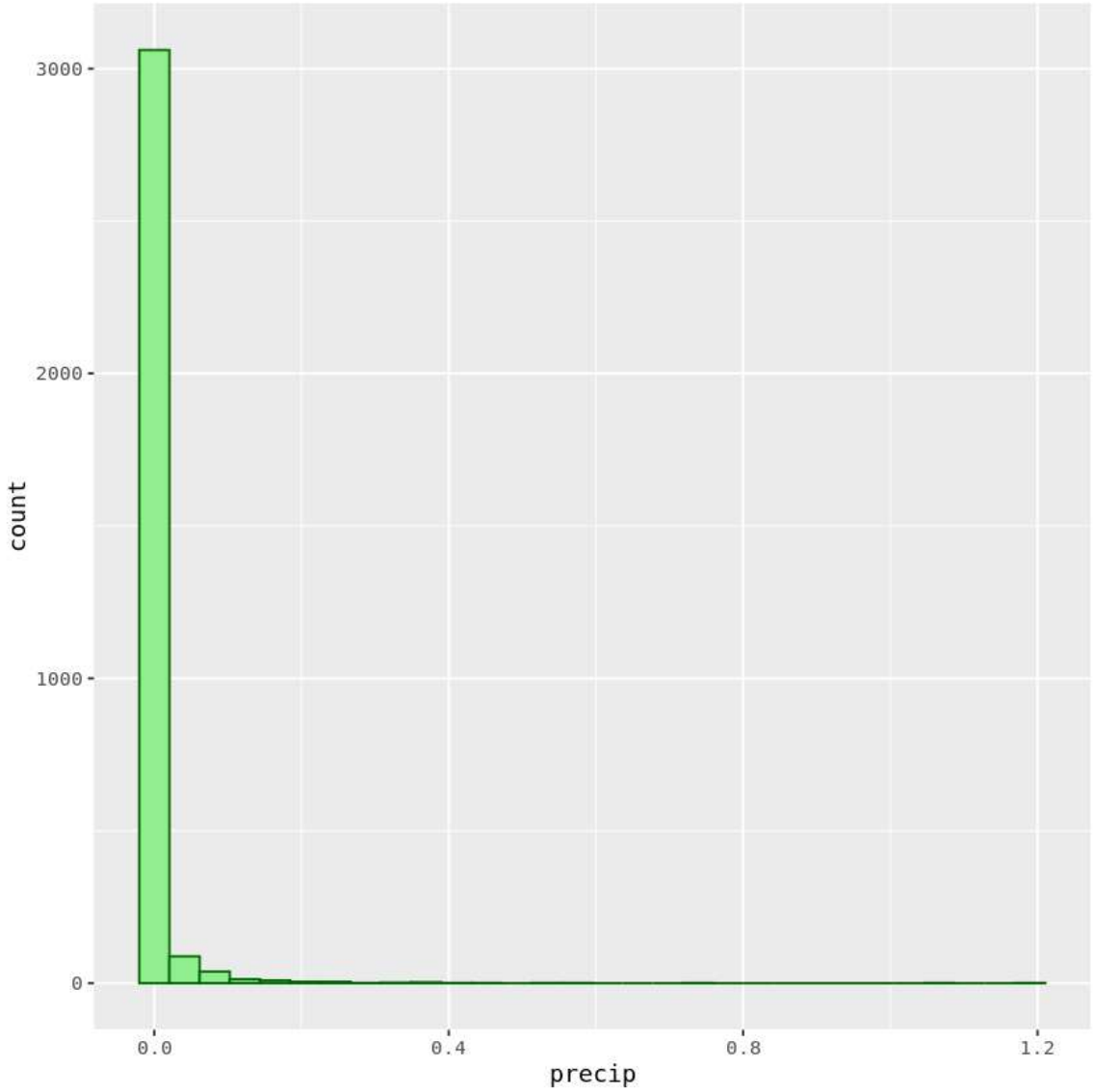
In [62]:
```
ggplot(train_set, aes(x = dry_bulb_temp_f))+
geom_histogram(color = " darkgreen", fill = "lightgreen")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.Warning message:
“Removed 114 rows containing non-finite values (`stat_bin()`).”
```



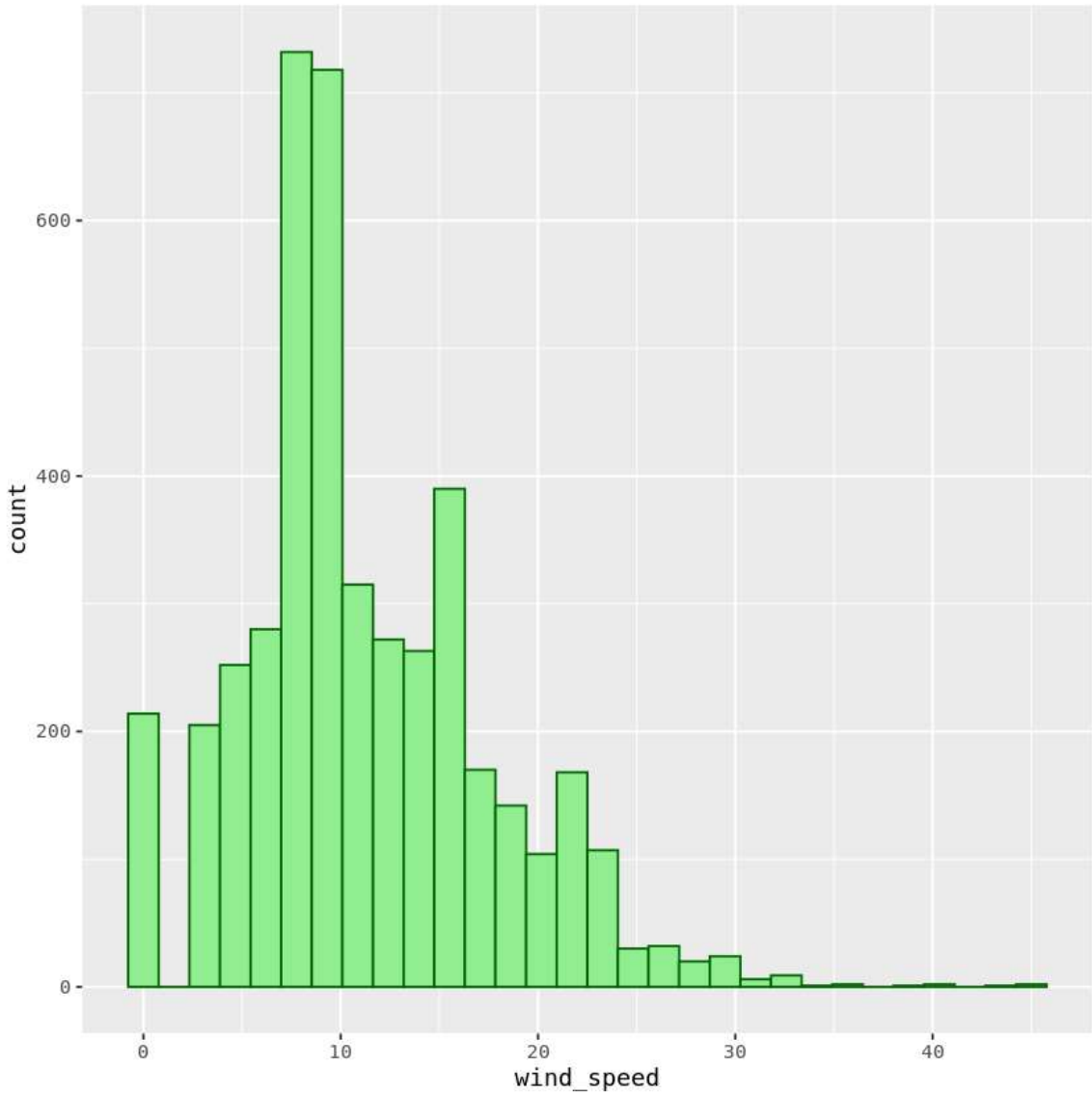
In [63]:
```
ggplot(train_set, aes(x = precip))+
geom_histogram(color = " darkgreen", fill = "lightgreen")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.Warning message:
“Removed 1349 rows containing non-finite values (`stat_bin()`).”
```



In [64]:
```
ggplot(train_set, aes(x = wind_speed))+
geom_histogram(color = " darkgreen", fill = "lightgreen")
```

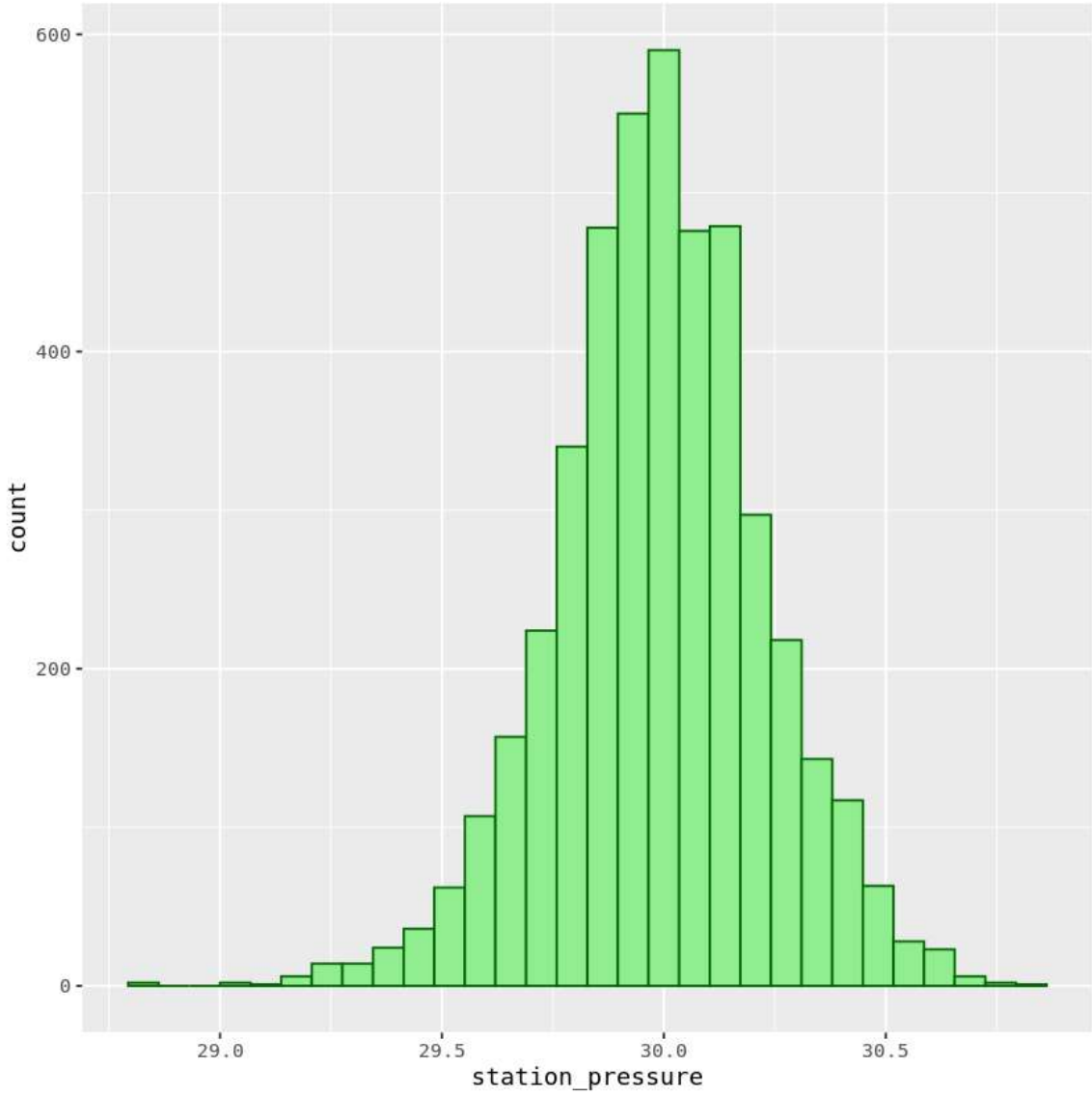
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.Warning message:
“Removed 119 rows containing non-finite values (`stat_bin()`).”
```



In [65]:

```
ggplot(train_set, aes(x = station_pressure))+
geom_histogram(color = " darkgreen", fill = "lightgreen")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.Warning message:
“Removed 121 rows containing non-finite values (`stat_bin()`).”
```



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 ~`

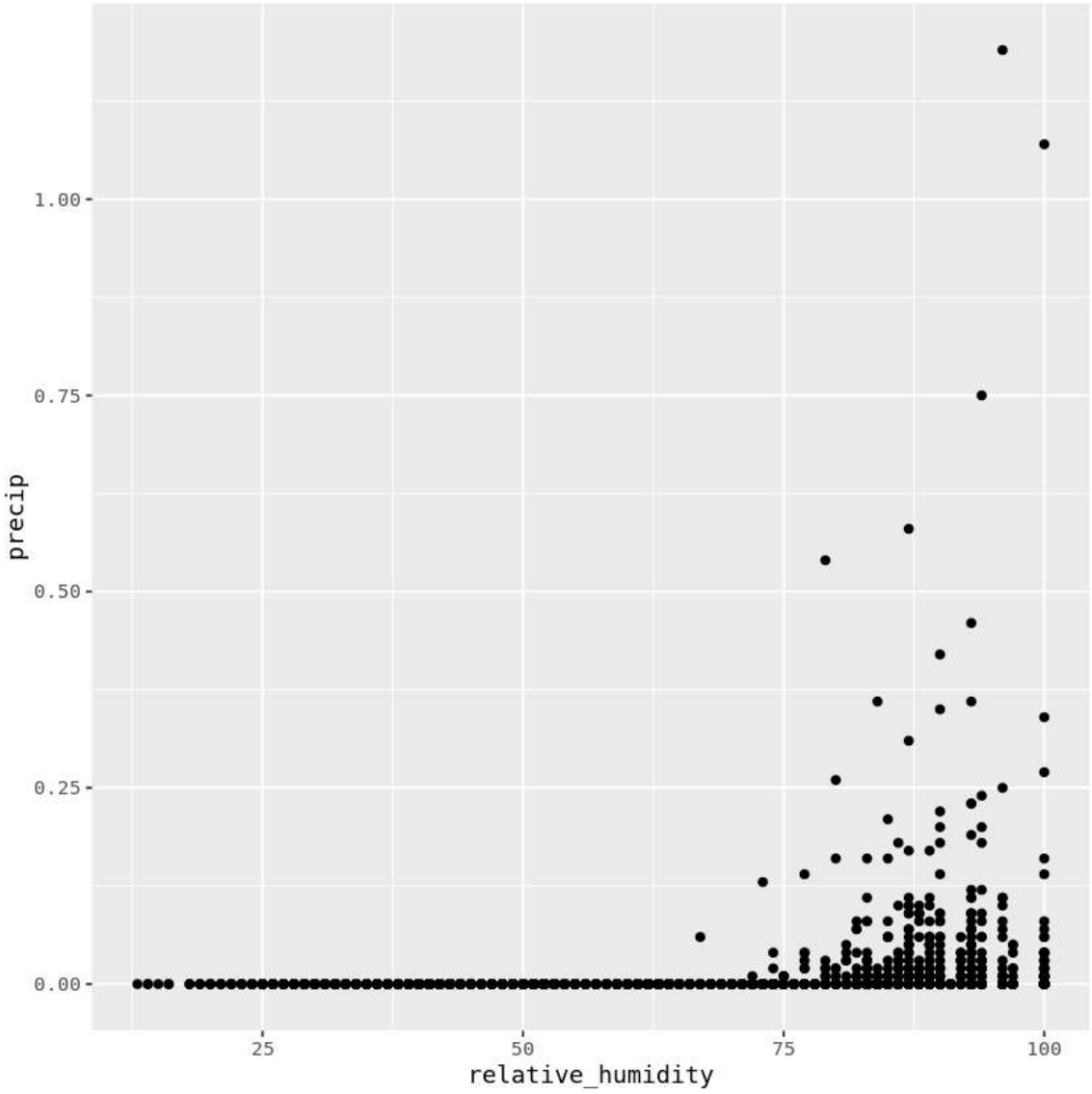
In [ ]:

In [83]:
```
# Simple linear regression
lm_a_train <- lm(precip ~ relative_humidity, data = train_set)
lm_b_train <- lm(precip ~ dry_bulb_temp_f, data = train_set)
lm_c_train <- lm(precip ~ wind_speed, data = train_set)
lm_d_train <- lm(precip ~ station_pressure, data = train_set)
```

In [ ]:

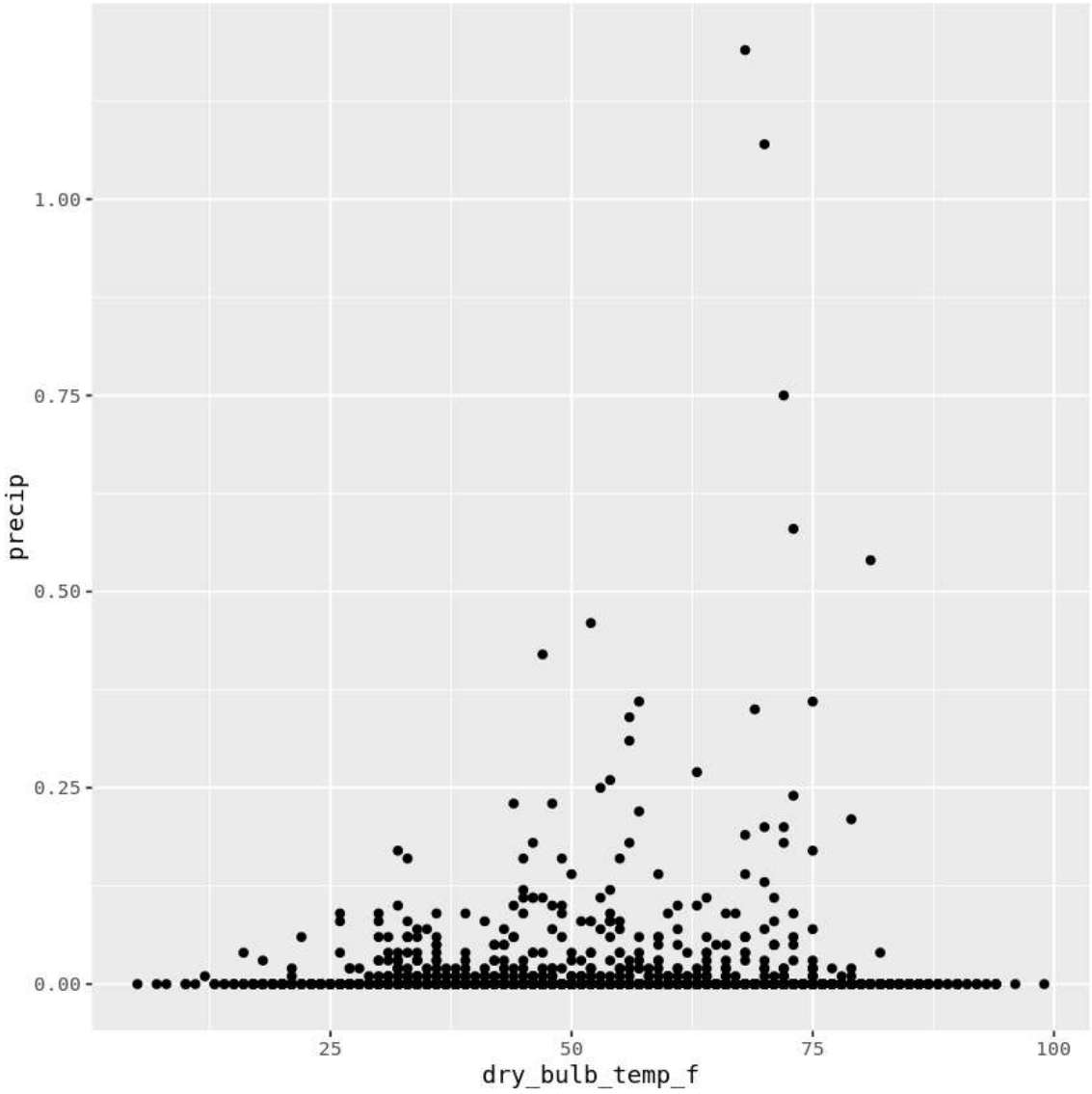
In [84]:
```
train_set %>%
ggplot(aes(relative_humidity, precip))+
  geom_point()
```

```
Warning message:
“Removed 1349 rows containing missing values (`geom_point()`).”
```



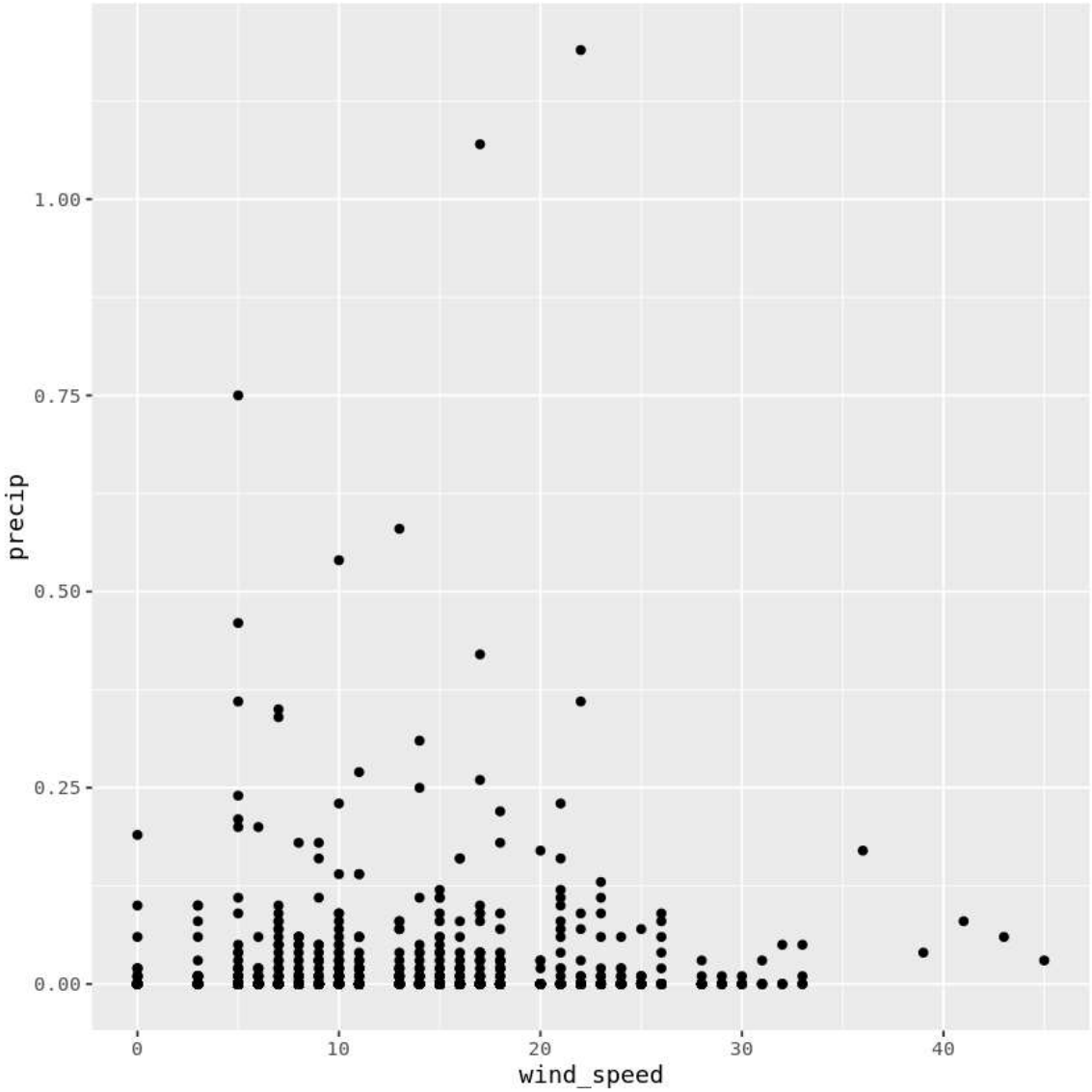
In [85]:
```
train_set %>%
ggplot(aes(dry_bulb_temp_f, precip))+
  geom_point()
```

```
Warning message:
“Removed 1349 rows containing missing values (`geom_point()`).”
```



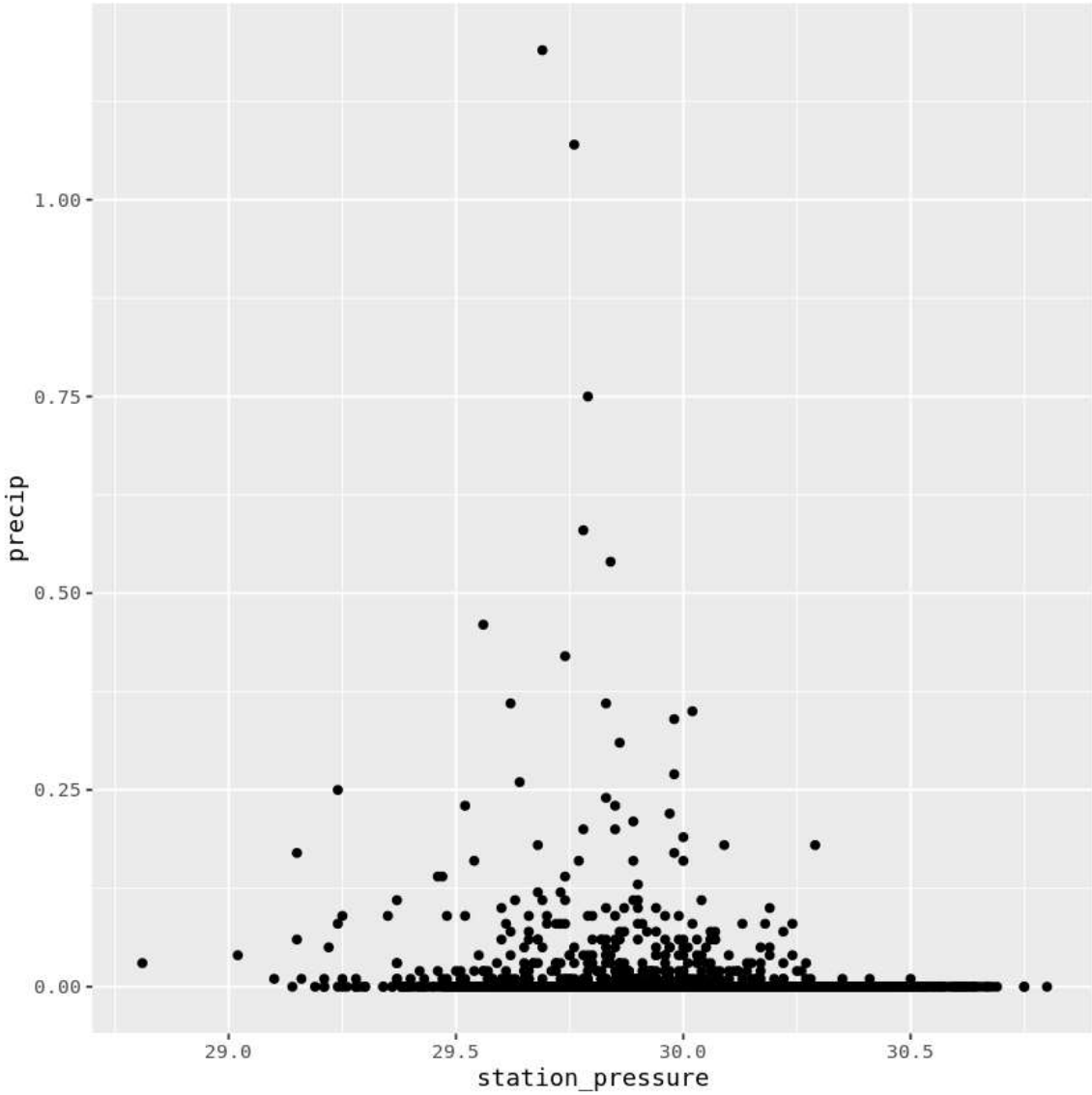
In [86]:
```
train_set %>%
ggplot(aes(wind_speed, precip))+
  geom_point()
```

```
Warning message:
“Removed 1349 rows containing missing values (`geom_point()`).”
```



In [96]:
```
train_set %>%
ggplot(aes(station_pressure, precip))+
  geom_point()
```

```
Warning message:
“Removed 1353 rows containing missing values (`geom_point()`).”
```



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

In [97]:
```
# Multiple linear regression
mlr_e_train <- lm(precip ~ relative_humidity + wind_speed, data = train_set)
mlr_f_train <- lm(precip ~ relative_humidity + wind_speed + station_pressure, data = train_set)
```

In [98]:
```
lm_a_test <- lm(precip ~ relative_humidity, data = test_set)
lm_b_test <- lm(precip ~ dry_bulb_temp_f, data = test_set)
lm_c_test <- lm(precip ~ wind_speed, data = test_set)
lm_d_test <- lm(precip ~ station_pressure, data = test_set)
```

In [99]:
```
mlr_e_test <- lm(precip ~ relative_humidity + wind_speed, data = test_set)
mlr_f_test <- lm(precip ~ relative_humidity + wind_speed + station_pressure, data = test_set)
```

In [113]:
```
#nova verzija, radi razlike u pred truth uzimala mean squared error po nizem principu (yt video)
#IDEJA JE kao da svaki put jedna varijabla utjecaja i jedan x i predikcija y-ona po njoj, pa pogreska predikcije po fitu po njoj, , jer zadatak
#da napravimo 4 SIMPLE LINEAR MODELA, IAKO OVO ZAPRAVO MULTIPLE LINEAR REGRESSION,  ONDA U IMPROVED DIJELU SAM RADILA KAO MULTIPLE LINEAR REG.
# A NAJBOLJI OD OVIH JE ONAJ KOJI IMA NAJMANJI MEAN SQUARED ERROR RECIMO, lm_a_train, b ,  c, d, e, f ;
```

In [114]:
```
lm_a_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_a_predict %>%
    fit(precip ~ relative_humidity , data = train_set)
#fit na train dijelu podataka

lm_a_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
#primjena predict train fita na test dijelu podataka i umetanje nove kolone predikt podataka i kolone stvarnih podataka precip da se vidi razlika predikcije i stvarnih podataka test set
#dataframe with new predicted data and real truth data for test set
#varijabla po varijabla,parametar po parametar ovdje
```

In [115]:
```
head(lm_a_results)
```

.pred	truth
-0.001561009	0
0.011883376	0
0.006179698	0
-0.005227660	0
0.008624131	NA
-0.005635065	NA

In [116]:
```
mse_lm_a_train <- mean(lm_a_train$residuals^2)
#razlika izmedju 2 stupca gore je pogrska i ko ima najmanju pogresku pobjednik od ovih predikcija
```

In [117]:
```
print(mse_lm_a_train)
```

```
[1] 0.001817648
```

In [118]:
```
#mogla sam i (https://www.youtube.com/watch?v=5MvyuMcqpFg)
#mse_lm_a_train <-((train_set$precip - predict(lm_a_train))^2)
```

In [119]:
```
lm_b_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_b_predict %>%
    fit(precip ~ dry_bulb_temp_f , data = train_set)

lm_b_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
```

In [120]:
```
head(lm_b_results)
```

.pred	truth
0.007429278	0
0.006074421	0
0.005526026	0
0.006171196	0
0.007042176	NA
0.006267972	NA

In [121]:
```
lm_c_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_c_predict %>%
    fit(precip ~ wind_speed , data = train_set)

lm_c_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
```

In [122]:
```
head(lm_c_results)
```

.pred	truth
0.007202385	0
0.004534512	0
0.006440136	0
0.004915637	0
0.004534512	NA
0.007202385	NA

In [123]:
```
lm_d_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_c_predict %>%
    fit(precip ~ station_pressure , data = train_set)

lm_d_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
```

In [124]:
```
head(lm_d_results)
```

.pred	truth
0.0066377203	0
0.0082196736	0
-0.0014980394	0
0.0007618938	0
0.0077676869	NA
0.0098016269	NA

In [125]:
```
lm_e_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_e_predict %>%
    fit(precip ~ relative_humidity + wind_speed , data = train_set)

lm_e_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
```

In [126]:
```
head(lm_e_results)
```

.pred	truth
-0.001383697	0
0.008722329	0
0.005935179	0
-0.009984160	0
0.005018439	NA
-0.006013559	NA

In [127]:
```
lm_f_predict <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_f_predict %>%
    fit(precip ~ relative_humidity + wind_speed + station_pressure , data = train_set)

lm_f_results <- train_fit %>%
  predict(new_data = test_set) %>%
  mutate(truth = test_set$precip)
```

In [128]:
```
head(lm_f_results)
```

.pred	truth
-0.0008009163	0
0.0096444696	0
0.0025868035	0
-0.0107881501	0
0.0060500859	NA
-0.0037019772	NA

In [ ]:
```
#od nize prva verzija
```

In [ ]:

In [129]:
```
mse_lm_a_train <- mean(lm_a_train$residuals^2)
rmse_lm_a_train <- sqrt(mse_lm_a_train)
rmse_lm_a_train

summary(lm_a_train)$r.squared
```

0.042633884986004

0.0348125959119728

In [130]:
```
mse_lm_b_train <- mean(lm_b_train$residuals^2)
rmse_lm_b_train <- sqrt(mse_lm_b_train)
rmse_lm_b_train

summary(lm_b_train)$r.squared
```

0.043392385033562

0.00016376925087888

In [131]:
```
mse_lm_c_train <- mean(lm_c_train$residuals^2)
rmse_lm_c_train <- sqrt(mse_lm_c_train)
rmse_lm_c_train

summary(lm_c_train)$r.squared
```

0.0433315518426125

0.00296520979252207

In [132]:
```
mse_lm_d_train <- mean(lm_d_train$residuals^2)
rmse_lm_d_train <- sqrt(mse_lm_d_train)
rmse_lm_d_train

summary(lm_d_train)$r.squared
```

0.0430772440922356

0.01557426038156

In [133]:
```
mse_mlr_e_train <- mean(mlr_e_train$residuals^2)
rmse_mlr_e_train <- sqrt(mse_mlr_e_train)
rmse_mlr_e_train

summary(mlr_e_train)$r.squared
```

0.0424013869180329

0.0453109273640681

In [134]:
```
mse_mlr_f_train <- mean(mlr_f_train$residuals^2)
rmse_mlr_f_train <- sqrt(mse_mlr_f_train)
rmse_mlr_f_train

summary(mlr_f_train)$r.squared
```

0.0423731277100182

0.047492986038494

In [135]:
```
mse_lm_a_test <- mean(lm_a_test$residuals^2)
rmse_lm_a_test <- sqrt(mse_lm_a_test)
rmse_lm_a_test

summary(lm_a_test)$r.squared
```

0.03322053062065

0.0587280705157021

In [136]:
```
mse_lm_b_test <- mean(lm_b_test$residuals^2)
rmse_lm_b_test <- sqrt(mse_lm_b_test)
rmse_lm_b_test

summary(lm_b_test)$r.squared
```

0.0342407570726119

2.60319245104704e-05

In [137]:
```
mse_lm_c_test <- mean(lm_c_test$residuals^2)
rmse_lm_c_test <- sqrt(mse_lm_c_test)
rmse_lm_c_test

summary(lm_c_test)$r.squared
```

0.0336190623854605

0.0360085732827907

In [138]:
```
mse_lm_d_test <- mean(lm_d_test$residuals^2)
rmse_lm_d_test <- sqrt(mse_lm_d_test)
rmse_lm_d_test

summary(lm_d_test)$r.squared
```

0.0336635239626001

0.0334571088498333

In [139]:
```
mse_mlr_e_test <- mean(mlr_e_test$residuals^2)
rmse_mlr_e_test <- sqrt(mse_mlr_e_test)
rmse_mlr_e_test

summary(mlr_e_test)$r.squared
```

0.0321504176207227

0.118392624992331

In [140]:
```
mse_mlr_f_test <- mean(mlr_f_test$residuals^2)
rmse_mlr_f_test <- sqrt(mse_mlr_f_test)
rmse_mlr_f_test

summary(mlr_f_test)$r.squared
```

0.0321171813867534

0.120214446648217

In [ ]:

In [ ]:

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.

In [ ]:

In [141]:
```
model_names <- c("lm_a", "lm_b", "lm_c", "lm_d", "mlr_e", "mlr_f")
train_rmse <- c("0.042633884986004", "0.043392385033562", "0.0433315518426125", "0.0430772440922356", "0.0424013869180329", "0.0423731277100182")
test_rmse <- c("0.03322053062065", "0.0342407570726119", "0.0336190623854605", "0.0336635239626001", "0.0321504176207227", "0.0321171813867534")
train_rsquared <- c("0.0348125959119728", "0.00016376925087888", "0.00296520979252207", "0.01557426038156", "0.0453109273640681", "0.047492986038494")
test_rsquared <- c("0.0587280705157021", "0.0000260319245104704", "0.0360085732827907", "0.0334571088498333", "0.118392624992331", "0.120214446648217")
comparison_df <- data.frame(model_names, train_rmse, test_rmse, train_rsquared, test_rsquared)
```

In [142]:
```
comparison_df %>%
#Comparison data frame is arrnaged by descending R-Squared values and ascending RMSE values
arrange(desc(train_rsquared), train_rmse)
#Based on the outcome, the mlr_2 offers the best model since the model has the highest R Square and the lowest RMSE.
```

model_names	train_rmse	test_rmse	train_rsquared	test_rsquared
mlr_f	0.0423731277100182	0.0321171813867534	0.047492986038494	0.120214446648217
mlr_e	0.0424013869180329	0.0321504176207227	0.0453109273640681	0.118392624992331
lm_a	0.042633884986004	0.03322053062065	0.0348125959119728	0.0587280705157021
lm_d	0.0430772440922356	0.0336635239626001	0.01557426038156	0.0334571088498333
lm_c	0.0433315518426125	0.0336190623854605	0.00296520979252207	0.0360085732827907
lm_b	0.043392385033562	0.0342407570726119	0.00016376925087888	0.0000260319245104704

Author(s)

Yiwen Li

Contributions

Tiffany Zhu

© IBM Corporation 2021. All rights reserved.

In [ ]: