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.

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 required modules

Tidymodels is a collection of packages that use tidyverse principles to easily do the entire modeling process from preprocessing initial data, to creating a model, to tunning hyperparameters. The tidymodels packages can be used to produce high quality statistical and machine learning models. Our Jupyter notebook platforms have a built-in Tidyverse, Tidymodels and rlang packages so we do not need to install these packages prior to loading library. However, if you decide to run this lab on your RStudio Desktop locally on your machine, you can remove the commented lines of code to install these packages before loading.

In [1]:

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

```
Installing package into ‘/opt/conda/envs/R-4.2-rt23.1/lib/R/extra-library’
(as ‘lib’ is unspecified)

Installing package into ‘/opt/conda/envs/R-4.2-rt23.1/lib/R/extra-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘tzdb’, ‘cpp11’, ‘warp’, ‘lhs’, ‘DiceDesign’, ‘lifecycl
e’, ‘pillar’, ‘tidyselect’, ‘vctrs’, ‘patchwork’, ‘clock’, ‘furrr’, ‘slider’, ‘stringr’,
‘GPfit’, ‘modelenv’, ‘broom’, ‘cli’, ‘conflicted’, ‘dials’, ‘dplyr’, ‘ggplot2’, ‘hardha
t’, ‘infer’, ‘modeldata’, ‘parsnip’, ‘purrr’, ‘recipes’, ‘rsample’, ‘rstudioapi’, ‘tibbl
e’, ‘tidyr’, ‘tune’, ‘workflows’, ‘workflowsets’, ‘yardstick’
```

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 [1]:

```
# Library for modeling
library(tidymodels)

# Load tidyverse
library(tidyverse)
```

```
── Attaching packages ──────────────────────────────────── tidymodels 1.1.1 ──

□ broom        1.0.5     □ recipes      1.0.8
□ dials        1.2.0     □ rsample      1.2.0
□ dplyr        1.1.4     □ tibble       3.2.1
□ ggplot2      3.4.4     □ tidyr        1.3.0
□ infer        1.0.5     □ tune         1.1.2
□ modeldata    1.2.0     □ workflows    1.1.3
□ parsnip      1.1.1     □ workflowsets 1.0.1
□ purrr        1.0.2     □ yardstick    1.2.0

── Conflicts ─────────────────────────────────────── tidymodels_conflicts() ──
```

```
□ purrr::discard() masks scales::discard()
□ dplyr::filter()  masks stats::filter()
□ dplyr::lag()     masks stats::lag()
□ recipes::step()  masks stats::step()
• Search for functions across packages at https://www.tidymodels.org/find/

── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

□ readr   2.1.2     □ forcats 0.5.1
□ stringr 1.5.1     

── 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()
```

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.

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'

In [8]:

```r
url <- 'https://dax-cdn.cdn.appdomain.cloud/dax-noaa-weather-data-jfk-airport/1.1.4/noaa

download.file(url, destfile = "noaa-weather-sample-data.tar.gz")
```

Untar the zipped file.

In [9]:

```
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 [11]:

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

```
Rows: 5727 Columns: 9
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (1): HOURLYPrecip
dbl  (7): HOURLYDewPointTempF, HOURLYRelativeHumidity, HOURLYDRYBULBTEMPF, H...
dttm (1): DATE

□ Use `spec()` to retrieve the full column specification for this data.
□ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Next, display the first few rows of the dataframe.

In [12]:

```
head(noaa_data)
```

A tibble: 6 × 9

DATE	HOURLYDewPointTempF	HOURLYRelativeHumidity	HOURLYDRYBULBTEMPF	HOURLYWETBULBTEMPF
<dttm>	<dbl>	<dbl>	<dbl>	<dbl>
2015-07-25 13:51:00	60	46	83	68
2016-11-18 23:51:00	34	48	53	44
2013-01-06 08:51:00	33	89	36	35
2011-01-27 16:51:00	18	48	36	30
2015-01-03 12:16:00	27	61	39	34
2013-02-15 20:51:00	35	79	41	38

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 [13]:

```
glimpse(noaa_data)
```

```
Rows: 5,727
Columns: 9
$ DATE                    <dttm> 2015-07-25 13:51:00, 2016-11-18 23:51:00, 2013…
$ HOURLYDewPointTempF     <dbl> 60, 34, 33, 18, 27, 35, 4, 14, 51, 71, 76, 19, …
$ 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,…
$ HOURLYWETBULBTEMPF     <dbl> 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        <dbl> 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 [15]:

```r
hrly_data <- noaa_data %>%
    select(HOURLYRelativeHumidity, HOURLYDRYBULBTEMPF, HOURLYPrecip,
          HOURLYWindSpeed, HOURLYStationPressure)
```

Show the first 10 rows of this new dataframe.

In [16]:

```r
head(hrly_data, 10)
```

A tibble: 10 × 5

HOURLYRelativeHumidity	HOURLYDRYBULBTEMPF	HOURLYPrecip	HOURLYWindSpeed	HOURLYStationPressure
<dbl>	<dbl>	<chr>	<dbl>	<dbl>
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 [17]:

```
unique(hrly_data$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'
```

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.

In [19]:

```
noaa_data$HOURLYPrecip <- replace(noaa_data$HOURLYPrecip, jfk_weather$HOURLYPrecip == "T
noaa_data$HOURLYPrecip <- str_remove(noaa_data$HOURLYPrecip, pattern = "s$")

unique(noaa_data$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'

In [20]:

```
sapply(noaa_data, typeof)
```

DATE: 'double' **HOURLYDewPointTempF:** 'double' **HOURLYRelativeHumidity:** 'double' **HOURLYDRYBULBTEMPF:** 'double' **HOURLYWETBULBTEMPF:** 'double' **HOURLYPrecip:** 'character' **HOURLYWindSpeed:** 'double' **HOURLYSeaLevelPressure:** 'double' **HOURLYStationPressure:** 'double'

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 [21]:

```
precip_data <- noaa_data %>%
    mutate(HOURLYPrecip = as.double(HOURLYPrecip))
```

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

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

In [22]:

```
glimpse(precip_data)
```

```
Rows: 5,727
Columns: 9
$ DATE                   <dttm> 2015-07-25 13:51:00, 2016-11-18 23:51:00, 2013…
$ HOURLYDewPointTempF    <dbl> 60, 34, 33, 18, 27, 35, 4, 14, 51, 71, 76, 19, …
$ 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,…
$ HOURLYWETBULBTEMPF     <dbl> 68, 44, 35, 30, 34, 38, 15, 21, 52, 72, 78, 35,…
$ 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, …
$ 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…
```

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 [23]:

```
names(precip_data) <- c('relative_humidity', 'dry_bulb_temp_f', 'precip', 'wind_speed',
head(precip_data)
```

```
Warning message:
“The `value` argument of `names<-()` must have the same length as `x` as of
tibble 3.0.0.”
Warning message:
“The `value` argument of `names<-()` can't be empty as of tibble 3.0.0.”
```

A tibble: 6 × 9

relative_humidity	dry_bulb_temp_f	precip	wind_speed	station_pressure	NA	NA	NA	NA
<dttm>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2015-07-25 13:51:00	60	46	83	68	0	13	30.01	29.99
2016-11-18 23:51:00	34	48	53	44	0	6	30.05	30.03
2013-01-06 08:51:00	33	89	36	35	0	13	30.14	30.12
2011-01-27 16:51:00	18	48	36	30	0	14	29.82	29.80
2015-01-03 12:16:00	27	61	39	34	0	11	NA	30.50
2013-02-15 20:51:00	35	79	41	38	0	6	29.94	29.92

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

In [32]:
```
summary(train_data_clean)
```

```
 relative_humidity              dry_bulb_temp_f      precip      
 Min.   :2010-01-01 04:51:00   Min.   :-14.00   Min.   : 13.00  
 1st Qu.:2012-04-06 15:51:00   1st Qu.: 28.00   1st Qu.: 49.00  
 Median :2014-05-01 22:51:00   Median : 45.00   Median : 65.00  
 Mean   :2014-04-27 12:20:25   Mean   : 42.79   Mean   : 64.52  
 3rd Qu.:2016-06-15 16:51:00   3rd Qu.: 59.00   3rd Qu.: 81.00  
 Max.   :2018-07-26 10:51:00   Max.   : 78.00   Max.   :100.00  
   wind_speed    station_pressure       NA                 NA       
 Min.   : 5.00   Min.   : 3.00    Min.   :0.000000   Min.   : 0.00  
 1st Qu.:42.00   1st Qu.:37.00    1st Qu.:0.000000   1st Qu.: 7.00  
 Median :56.00   Median :50.00    Median :0.000000   Median :10.00  
 Mean   :55.82   Mean   :49.71    Mean   :0.004636   Mean   :11.14  
 3rd Qu.:70.00   3rd Qu.:64.00    3rd Qu.:0.000000   3rd Qu.:15.00  
 Max.   :99.00   Max.   :80.00    Max.   :1.190000   Max.   :45.00  
       NA              NA       
 Min.   :28.83   Min.   :28.81  
 1st Qu.:29.88   1st Qu.:29.86  
 Median :30.02   Median :30.00  
 Mean   :30.03   Mean   :30.01  
 3rd Qu.:30.17   3rd Qu.:30.15  
 Max.   :30.82   Max.   :30.80  
```

In [30]:
```
# Remove rows with NA, NaN, or Inf values
train_data_clean <- na.omit(train_data)
```

In [35]:
```
# Check for columns with NA or empty names
cols_to_keep <- names(train_data_clean)[!is.na(names(train_data_clean)) & names(train_da
# Filter the dataset to keep columns without NA or empty names
cleaned_train_data <- train_data_clean[, cols_to_keep]
```

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.

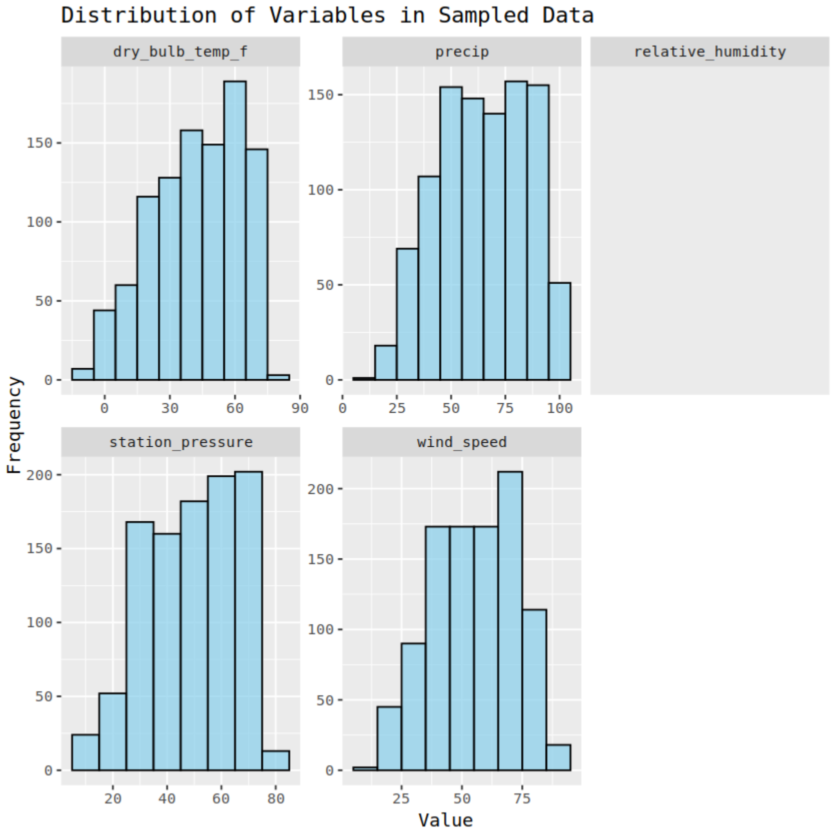
In [45]:
```
library(ggplot2)
library(tidyr)
library(dplyr)

# Selecting specific variables for visualization
selected_vars <- c("relative_humidity", "dry_bulb_temp_f", "precip", "wind_speed", "stat

# Sample 10,000 rows from the cleaned dataset for visualization
set.seed(123)  # Setting a seed for reproducibility
sampled_data <- cleaned_train_data %>%
  select(all_of(selected_vars)) %>%
  sample_n(1000, replace = TRUE)

# Creating histograms for the selected variables
sampled_data %>%
  gather(cols, value) %>%
  ggplot(aes(x = value)) +
  geom_histogram(binwidth = 10, color = "black", fill = "skyblue", alpha = 0.7) +
  facet_wrap(~cols, scales = "free") +
  labs(title = "Distribution of Variables in Sampled Data", x = "Value", y = "Frequency"
```

```
Warning message:
“attributes are not identical across measure variables; they will be dropped”
Warning message:
“Computation failed in `stat_bin()`
Caused by error in `bin_breaks_width()`:
! The number of histogram bins must be less than 1,000,000.
□ Did you make `binwidth` too small?”
```



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.

In [47]:

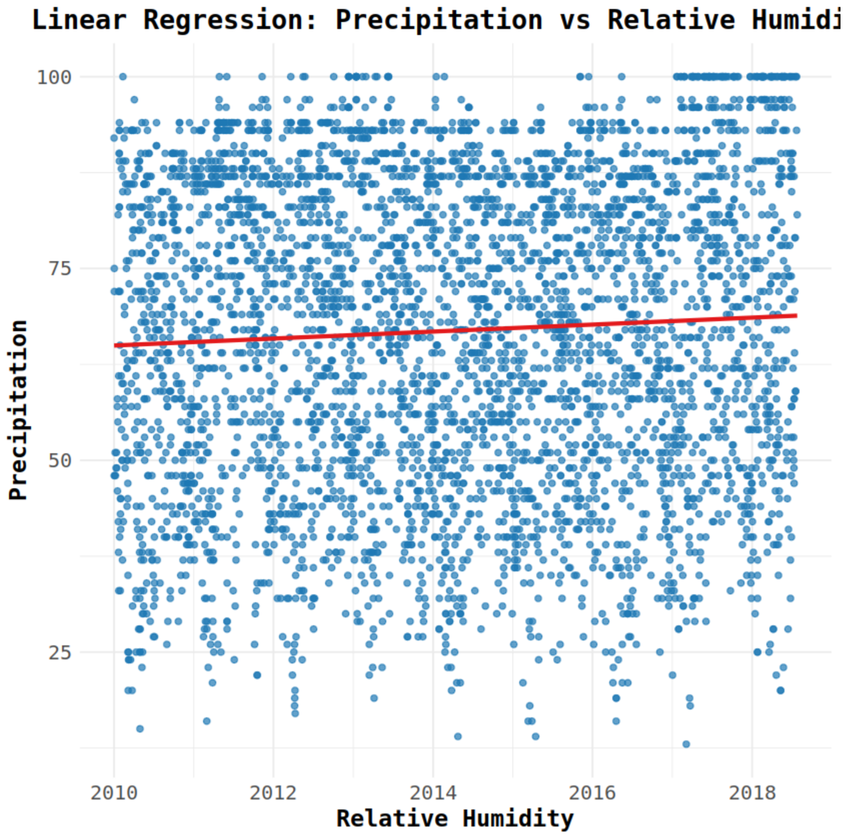
```r
library(tidymodels)
library(ggplot2)

# Define and fit the linear regression model
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_spec %>%
  fit(precip ~ relative_humidity, data = train_data)

# Visualize the relationship between relative humidity and precipitation
ggplot(train_data, aes(x = relative_humidity, y = precip)) +
  geom_point(alpha = 0.7, color = "#1F78B4") +  # Adjusting alpha and point color
  geom_smooth(method = "lm", formula = y ~ x, color = "#E31A1C", se = FALSE, size = 1.2)
  labs(
    title = "Linear Regression: Precipitation vs Relative Humidity",
    x = "Relative Humidity",
    y = "Precipitation"
  ) +
  theme_minimal() +  # Using a minimal theme for clarity
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),  # Adjusting title
    axis.text = element_text(size = 12),  # Modifying axis text size
    axis.title = element_text(size = 14, face = "bold")  # Adjusting axis title appearan
  )
```

```
Warning message:
“Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
□ Please use `linewidth` instead.”
Warning message:
“Removed 114 rows containing non-finite values (`stat_smooth()`).”
Warning message:
“Removed 114 rows containing missing values (`geom_point()`).”
```



In [48]:

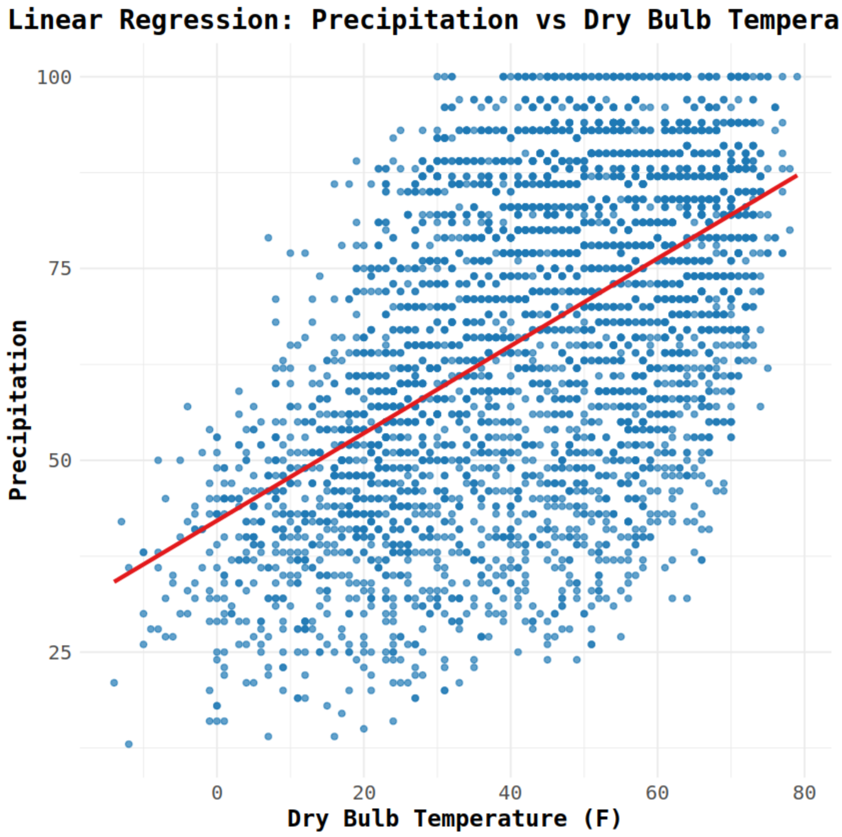
```
library(tidymodels)
library(ggplot2)

# Define and fit the linear regression model
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_spec %>%
  fit(precip ~ dry_bulb_temp_f, data = train_data)

# Visualize the relationship between dry bulb temperature and precipitation
ggplot(train_data, aes(x = dry_bulb_temp_f, y = precip)) +
  geom_point(alpha = 0.7, color = "#1F78B4") +  # Adjusting alpha and point color
  geom_smooth(method = "lm", formula = y ~ x, color = "#E31A1C", se = FALSE, size = 1.2)
  labs(
    title = "Linear Regression: Precipitation vs Dry Bulb Temperature",
    x = "Dry Bulb Temperature (F)",
    y = "Precipitation"
  ) +
  theme_minimal() +  # Using a minimal theme for clarity
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),  # Adjusting title
    axis.text = element_text(size = 12),  # Modifying axis text size
    axis.title = element_text(size = 14, face = "bold")  # Adjusting axis title appearan
  )
```

```
Warning message:
“Removed 114 rows containing non-finite values (`stat_smooth()`).”
Warning message:
“Removed 114 rows containing missing values (`geom_point()`).”
```



In [49]:

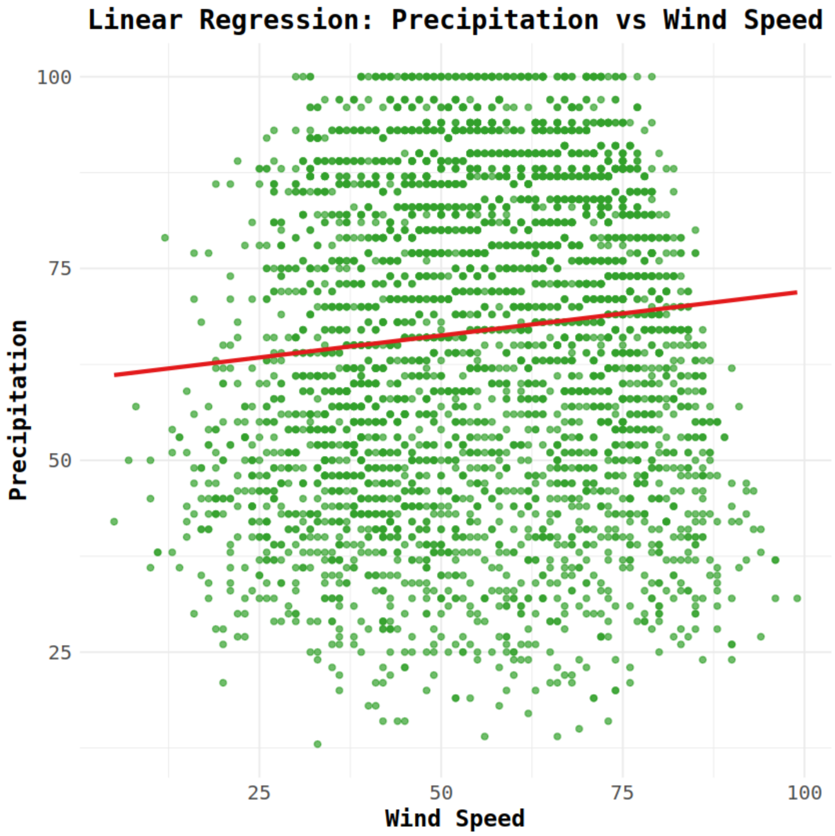
```
library(tidymodels)
library(ggplot2)

# Define and fit the linear regression model
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm")

train_fit <- lm_spec %>%
  fit(precip ~ wind_speed, data = train_data)

# Visualize the relationship between wind speed and precipitation
ggplot(train_data, aes(x = wind_speed, y = precip)) +
  geom_point(alpha = 0.7, color = "#33A02C") +  # Adjusting alpha and point color
  geom_smooth(method = "lm", formula = y ~ x, color = "#E31A1C", se = FALSE, size = 1.2)
  labs(
    title = "Linear Regression: Precipitation vs Wind Speed",
    x = "Wind Speed",
    y = "Precipitation"
  ) +
  theme_minimal() +  # Using a minimal theme for clarity
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),  # Adjusting title
    axis.text = element_text(size = 12),  # Modifying axis text size
    axis.title = element_text(size = 14, face = "bold")  # Adjusting axis title appearan
  )
```

```
Warning message:
“Removed 114 rows containing non-finite values (`stat_smooth()`).”
Warning message:
“Removed 114 rows containing missing values (`geom_point()`).”
```



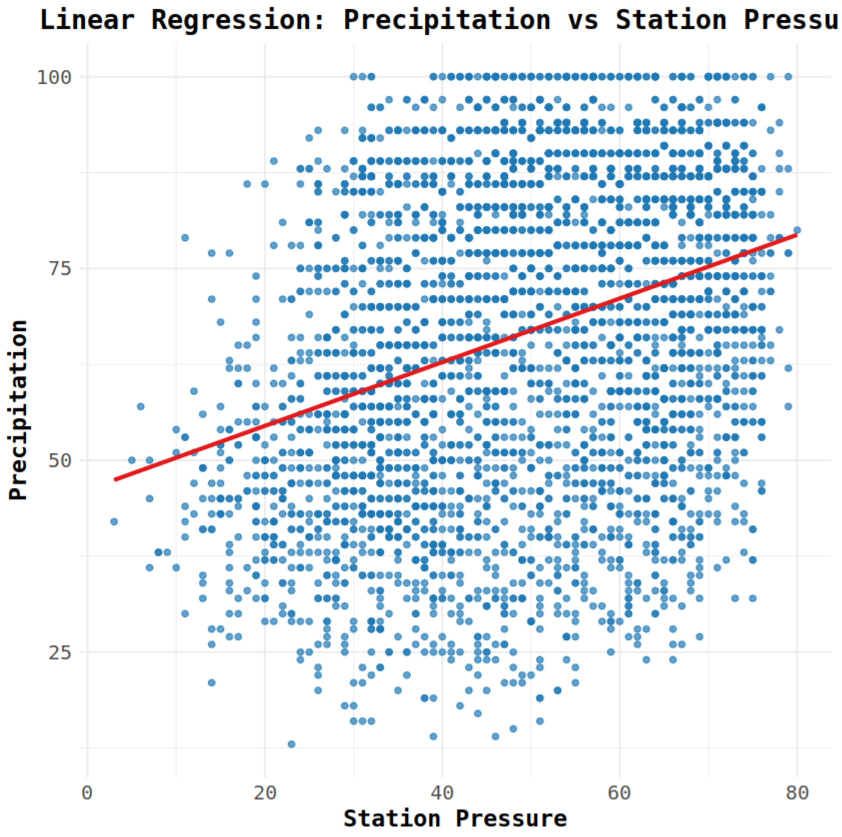
```
In [50]: library(tidymodels)
         library(ggplot2)

         # Define and fit the linear regression model
         lm_spec <- linear_reg() %>%
           set_engine(engine = "lm")

         train_fit <- lm_spec %>%
           fit(precip ~ station_pressure, data = train_data)

         # Visualize the relationship between station pressure and precipitation
         ggplot(train_data, aes(x = station_pressure, y = precip)) +
           geom_point(alpha = 0.7, color = "#1F78B4") +  # Adjusting alpha and point color
           geom_smooth(method = "lm", formula = y ~ x, color = "#E31A1C", se = FALSE, size = 1.2)
           labs(
             title = "Linear Regression: Precipitation vs Station Pressure",
             x = "Station Pressure",
             y = "Precipitation"
           ) +
           theme_minimal() +  # Using a minimal theme for clarity
           theme(
             plot.title = element_text(size = 16, face = "bold", hjust = 0.5),  # Adjusting title
             axis.text = element_text(size = 12),  # Modifying axis text size
             axis.title = element_text(size = 14, face = "bold")  # Adjusting axis title appearan
           )
```

```
Warning message:
“Removed 117 rows containing non-finite values (`stat_smooth()`).”
Warning message:
“Removed 117 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 [52]:

```
# Convert relevant columns to numeric format
train_data$relative_humidity <- as.numeric(train_data$relative_humidity)
# Repeat for other columns as needed (e.g., dry_bulb_temp_f, wind_speed)

# Fit the polynomial regression
poly_percip <- lm_spec %>%
  fit(precip ~ poly(relative_humidity, 2, raw = TRUE), data = train_data)

res_train <- poly_percip %>%
            predict(new_data = train_data) %>%
            mutate(truth = train_data$precip)
train_rmse <- rmse(res_train, truth = truth, estimate = .pred)
train_rmse
#mlr
mlr_train <- lm_spec %>% fit(precip ~ relative_humidity + dry_bulb_temp_f + wind_speed,
mlr_res <- mlr_train %>%
            predict(new_data = train_data) %>%
            mutate(truth = train_data$precip)
```

```
mlr_rmse <- rmse(mlr_res, truth = truth, estimate = .pred)
mlr_rmse
```

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	19.97166

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	3.648825

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 [59]:

```
# Predict on testing set for each model

# Convert relative_humidity to numeric explicitly
train_data$relative_humidity <- as.numeric(train_data$relative_humidity)
test_data$relative_humidity <- as.numeric(test_data$relative_humidity)

# Then, fit the polynomial regression model
poly_percip <- lm_spec %>% fit(precip ~ poly(relative_humidity, 2, raw = TRUE), data = t

# Proceed with prediction and evaluation
# ...

poly_pred_test <- poly_percip %>%
  predict(new_data = test_data) %>%
  mutate(truth = test_data$precip)

mlr_pred_test <- mlr_train %>%
  predict(new_data = test_data) %>%
  mutate(truth = test_data$precip)

# Calculate evaluation metrics on testing set
# Assuming you have functions to calculate metrics like RMSE
poly_test_rmse <- rmse(poly_pred_test, truth = truth, estimate = .pred)
mlr_test_rmse <- rmse(mlr_pred_test, truth = truth, estimate = .pred)

# Create a comparison table
model_names <- c("Poly Model", "MLR Model")  # Update with actual model names
train_error <- c("poly_train_rmse", "mlr_train_rmse")  # Use actual training set metrics
test_error <- c(poly_test_rmse, mlr_test_rmse)
```

```
# Create a dataframe for comparison
comparison_df <- data.frame(model_names, train_error, test_error)
```

In [60]:

```
comparison_df
```

A data.frame: 2 × 8

model_names	train_error	.metric	.estimator	.estimate	.metric.1	.estimator.1	.estimate.1
<chr>	<chr>	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
Poly Model	poly_train_rmse	rmse	standard	20.03757	rmse	standard	3.403041
MLR Model	mlr_train_rmse	rmse	standard	20.03757	rmse	standard	3.403041

Author(s)

Yiwen Li

Contributions

Tiffany Zhu