Predicting the Water Temperature off the South Coast

Application of ML Models on Santa Barbara Channel NOAA Weather Data

Kaden Nichols

Introduction

The ultimate goal of this project is to build a Machine Learning model that predicts the water temperature off the South Coast of Santa Barbara.

Inspiration

Growing up in Southern California, the Pacific Ocean has been a powerful influence on my daily life. Beyond shaping global weather patterns, it also directly and indirectly determines how much time I, and many others, spend outdoors.

However, the water along the central coast are far from tropical- and certainly not warm. On colder days, I often have to summon a little extra courage (or slip into a thicker wetsuit) just to venture into the ocean. There's nothing worse than looking at the ocean on a beautiful, sunny day only to dive in and immediately become a human popsicle.

This got me thinking: Is there a way to predict how cold the water will be just by looking at the beach? If I can anticipate those cold days, I may be better prepared to avoid those brutal ice cream headaches.

Data Description

For this project, I am using meteorological data from the National Oceanic and Atmospheric Administration's (NOAA) National Data Buoy Center (NDBC). Specifically, the data comes from Buoy LLNR 196, located 12 nautical miles southwest of Santa Barbara, CA. Due to its proximity to UCSB and its strategic position in the Santa Barbara Channel, LLNR 196 provides the most relevant meteorological information for my predictions.

NOAA is globally recognized as a reliable source for meteorological data; employing cutting-edge technologies, it is vital in daily forecasts, disaster prediction, and more. The NDBC, NOAA's division responsible for real-time marine weather data collection, operates over 100 buoys and 50 coastal weather stations. NDBC's data supports research, oceanographic modeling, and, for practical purposes, helps predict when the water will be unusually cold— like in this study.

The data set I am using for this project is the annual meteorological data collected by LLNR 196 over the entirety of 2022. In this project, I will focus only on on variables that directly affect the experience of being at the beach, such as swell, wind, and air temperature, among others.

The raw data set contains over 50000 observations across 13 different weather related measurements. Some variables, such as Visibility and Dewpoint Temperature, are irrelevant to this model and will be excluded. I will explain the selection of relevant variables in a later section.

Variable	Description
YY	*Year*
MM	*Month*
DD	*Day*
hh	*Hour(00:00)*
mm	*Minute*
WDIR	*Wind direction in degrees clockwise from North*
WSPD	*Wind speed (m/s) over 8 minute period*
GST	*Max Wind gust speed measured over same period as WSPD*
WVHT	*Significant Wave height (m) taken from average of highest 1/3 wave heights during measurement period*
DPD	*Dominant wave period (s) of maximum wave energy*
APD	*Average wave period (s) ob waves during observation period*
MWD	*Direction from which dominant waves are coming, measured with 0 as true north, 90 east*
PRES	*Sea level pressure (hPA)*
ATMP	*Air temperature (Celsius)*
WTMP	*Sea surface temperature (Celsius)*
DEWP	*Dewpoint temperature (Celsius)*
VIS	*Visibility from station (NM)*
PTDY	*Pressure tendency is the direction, amount of pressure change ending at the observation time*
TIDE	*Water level (feet) above or below Mean Lower Low Water*

Data Preprocessing

Loading Our Data

Let's begin by loading the raw data set, and examine its summary.

```
##      X.YY              MM               DD               hh
##  Length:52481       Length:52481       Length:52481       Length:52481
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##       mm                WDI               R.WSP             D.GST
##  Length:52481       Length:52481       Length:52481       Length:52481
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      WVHT              DPD               APD               MWD
##  Length:52481       Length:52481       Length:52481       Length:52481
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      PRES              ATMP              WTMP              DEWP
##  Length:52481       Length:52481       Length:52481       Length:52481
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      VIS               TIDE
##  Length:52481       Length:52481
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

As the summary demonstrates, we need to convert the data types for all of our columns from character to numeric format. This will better allow us to perform initial visualizations and facilitate the cleaning of the data. Further refining the data set will also enable us to make more accurate analyses as we proceed in building our model.

Initial Cleaning and Preprocessing

```
sb2022[] = sapply(sb2022, as.numeric) # changing to numeric types
```

The columns VIS, TIDE, DEWP, PRES, D.GST, and APD will also not be relevant in this project.

```
to_drop <- c("VIS", "TIDE", "DEWP", "D.GST", "APD", "PRES")
sb2022 = sb2022[,!(names(sb2022) %in% to_drop)]
```

Additionally, we should combine the columns of the Year, Month, Day, Hour, and Minute into a single column of date type. Once converted to a date format we can remove the original columns and simplify our data set.

Making an Appropriate Date Column

```
sb2022$Date <- ymd_hm(paste0(sb2022$'X.YY', sprintf("%02d", sb2022$MM), sprintf("%02d",
sb2022$DD), sprintf("%02d", sb2022$hh), sprintf("%02d", sb2022$mm)))
```

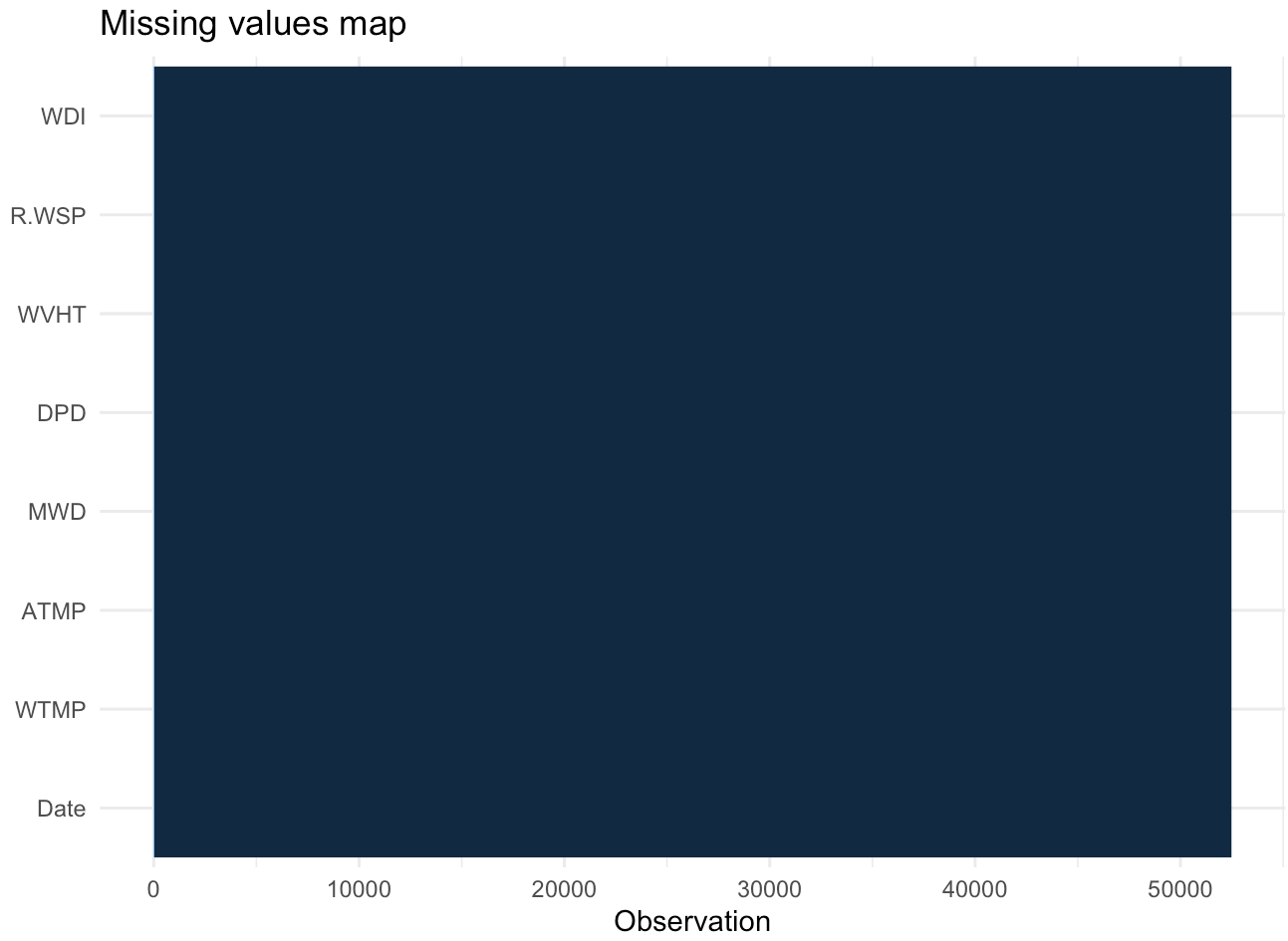
```
## Warning: 1 failed to parse.
```

```
drop_date <- c("X.YY", "MM", "DD", "hh", "mm")
sb2022 = sb2022[,!(names(sb2022) %in% drop_date)]
```

Missing Values

With this reduced data set, the next step is to check for missing values. Addressing the missing data will serve to ensure the accuracy and reliability of our model and analyses. In identifying gaps in our data, we can take appropriate measures to best maintain the integrity of our data set.

```
sb2022 %>% missing_plot()
```




```
sum(is.na(sb2022))
```

```
## [1] 8
```

It is difficult to count the missing observations from our missing plot, but there is only 8 missing data points for about 50000 observations. Thus, we can safely omit such cases with little impact on the integrity of the data set.

```
sb2022 = sb2022 %>% na.omit() # omit NA observations
```

With the missing data now omitted, we can begin identifying potential outliers of our data using the summary of our further refined data set. Identifying outliers is critical in ensuring the reliability of our data as we move forward with our analyses.

```
summary(sb2022)
```

```
##       WDI             R.WSP             WVHT            DPD
##  Min.   :  1.0   Min.   : 0.000   Min.   : 0.31   Min.   : 2.60
##  1st Qu.:216.0   1st Qu.: 2.100   1st Qu.:99.00   1st Qu.:99.00
##  Median :254.0   Median : 3.800   Median :99.00   Median :99.00
##  Mean   :230.4   Mean   : 4.324   Mean   :80.52   Mean   :82.24
##  3rd Qu.:277.0   3rd Qu.: 6.000   3rd Qu.:99.00   3rd Qu.:99.00
##  Max.   :999.0   Max.   :99.000   Max.   :99.00   Max.   :99.00
##       MWD              ATMP             WTMP
##  Min.   :  0.0   Min.   :  7.80   Min.   : 11.10
##  1st Qu.:999.0   1st Qu.: 13.60   1st Qu.: 14.00
##  Median :999.0   Median : 14.70   Median : 15.00
##  Mean   :860.1   Mean   : 16.86   Mean   : 21.85
##  3rd Qu.:999.0   3rd Qu.: 16.50   3rd Qu.: 17.30
##  Max.   :999.0   Max.   :999.00   Max.   :999.00
##       Date
##  Min.   :2022-01-01 00:00:00.00
##  1st Qu.:2022-04-02 04:27:30.00
##  Median :2022-07-02 08:55:00.00
##  Mean   :2022-07-02 11:40:23.97
##  3rd Qu.:2022-10-01 21:12:30.00
##  Max.   :2022-12-31 23:50:00.00
```

Interestingly, there are outliers in each numeric column. For example, it makes no sense to have directions of 999 degrees from true north, or air and water temperature of 999 degrees F. Could these cases be attributed to instrumental error? Nonetheless, we need to better understand the nature of these outlying data points.

Dealing with Outliers

The Wave Height readings of 99 meters and Wind Direction readings of 999 degrees may indicate some flawed measurements or data entry errors. Here, we will see how many instances of each case are present in the data set.

Let's first look at the amount of values of Wave Height are 99 meters and how many observations of wind direction are 999.

```
sum(sb2022$WVHT == 99)
```

```
## [1] 42566
```

```
sum(sb2022$MWD == 999)
```

```
## [1] 42566
```

Interestingly, these extreme values each appear in 42566 rows of our data. Maybe they're all in the same rows? If that's the case, then we could reduce our data drastically by removing them all simultaneously.

```
sum(sb2022$MWD == 999 & sb2022$WVHT == 99)
```

```
## [1] 42566
```

It turns out that these outliers do exist in the same samples. Removing these samples entirely will further ensure the integrity of our analyses.

```
sb2022 <- sb2022[!(sb2022$WVHT == 99),  ]
dim(sb2022) # our data set is now only 9914 observations!
```

```
## [1] 9914    8
```

Let's refer back and see how removing these outliers affected our data set.

```
View(sb2022)
summary(sb2022)
```

```
##       WDI            R.WSP            WVHT            DPD             MWD
##  Min.   :  1.0   Min.   : 0.000   Min.   :0.310   Min.   : 2.60   Min.   :  0
##  1st Qu.:209.0   1st Qu.: 2.000   1st Qu.:0.850   1st Qu.: 7.14   1st Qu.:259
##  Median :254.0   Median : 3.600   Median :1.100   Median :10.00   Median :266
##  Mean   :228.3   Mean   : 4.152   Mean   :1.181   Mean   :10.31   Mean   :264
##  3rd Qu.:278.0   3rd Qu.: 5.800   3rd Qu.:1.450   3rd Qu.:12.90   3rd Qu.:272
##  Max.   :999.0   Max.   :99.000   Max.   :3.420   Max.   :21.05   Max.   :359
##       ATMP            WTMP            Date
##  Min.   :  8.10   Min.   : 11.20   Min.   :2022-01-01 00:40:00.00
##  1st Qu.: 13.60   1st Qu.: 14.10   1st Qu.:2022-04-14 21:55:00.00
##  Median : 14.60   Median : 15.00   Median :2022-07-28 00:10:00.00
##  Mean   : 16.07   Mean   : 20.73   Mean   :2022-07-22 09:09:52.92
##  3rd Qu.: 16.20   3rd Qu.: 16.90   3rd Qu.:2022-11-10 08:02:30.00
##  Max.   :999.00   Max.   :999.00   Max.   :2022-12-31 23:40:00.00
```

At this point we have identified more data points that still appear inconsistent, particularly in the WDI, R.WSP, WTMP and ATMP columns. These inconsistencies warrant further investigation.

```
sum(sb2022$WDI  == 999) # 1 observation
```

```
## [1] 1
```

```
sum(sb2022$R.WSP == 99) # 1 observation
```

```
## [1] 1
```

```
sum(sb2022$ATMP == 999) # 11 observations
```

```
## [1] 11
```

```
sum(sb2022$WTMP == 999) # 53 observations
```

```
## [1] 53
```

There are 66 more inconsistent samples in our data set; with a total of 9914 observations, removing such erroneous samples should not affect the integrity of our data.

```
sb2022 <- sb2022[!(sb2022$WTMP == 999),  ]
sb2022 <- sb2022[!(sb2022$ATMP == 999),  ]
sb2022 <- sb2022[!(sb2022$WDI == 999),  ]
sb2022 <- sb2022[!(sb2022$R.WSP == 99),  ]
```

Let's take another look at our data to evaluate omitting these 66 samples improves the overall quality of our data set.

```
##       WDI            R.WSP            WVHT            DPD            MWD
##  Min.   :  1.0   Min.   : 0.00   Min.   :0.310   Min.   : 2.60   Min.   :  0
##  1st Qu.:209.0   1st Qu.: 2.00   1st Qu.:0.850   1st Qu.: 7.14   1st Qu.:259
##  Median :254.0   Median : 3.60   Median :1.100   Median :10.00   Median :266
##  Mean   :228.3   Mean   : 4.14   Mean   :1.181   Mean   :10.31   Mean   :264
##  3rd Qu.:279.0   3rd Qu.: 5.80   3rd Qu.:1.450   3rd Qu.:12.90   3rd Qu.:272
##  Max.   :360.0   Max.   :14.80   Max.   :3.420   Max.   :21.05   Max.   :359
##       ATMP            WTMP            Date
##  Min.   : 8.10   Min.   :11.20   Min.   :2022-01-01 00:40:00.00
##  1st Qu.:13.60   1st Qu.:14.10   1st Qu.:2022-04-14 15:55:00.00
##  Median :14.60   Median :14.90   Median :2022-07-27 16:10:00.00
##  Mean   :14.98   Mean   :15.47   Mean   :2022-07-22 06:58:22.10
##  3rd Qu.:16.20   3rd Qu.:16.90   3rd Qu.:2022-11-10 09:02:30.00
##  Max.   :24.70   Max.   :22.30   Max.   :2022-12-31 23:40:00.00
```

The values of each column are now more appropriate within their respective units, but 9000 observations means our data set it still too large for efficiently constructing a machine learning model. To address this, I will reduce the data set size by averaging samples on a daily basis. Since the outlying data points have already been dealt with, this aggregation should provide an accurate and representative data set for our models.

Reducing the Length of our Data Set

```
dailysb22 <- sb2022 %>%
  group_by(date = as.Date(Date)) %>% # Group by date
  summarise(airtemp = mean(ATMP), # Take the average of each day across columns
            watertemp = mean(WTMP),
            wavedirection = mean(MWD),
            waveperiod = mean(DPD),
            waveheight = mean(WVHT),
            windspeed = mean(R.WSP),
            winddirection = mean(WDI))
dim(dailysb22) # we now have a data set with 365 rows!
```

```
## [1] 365   8
```

Next, we will need to transform some of the data into more familiar units: meters to feet, meters per second to knots, and Celsius to Fahrenheit.

Changing Units

```
# Changing Air/Water Temp to Fahrenheit
dailysb22 = dailysb22 %>% mutate(watertemp=((watertemp*(9/5))+32))
dailysb22 = dailysb22 %>% mutate(airtemp=((airtemp*(9/5))+32))
```

```
# Changing Wave Height from meters into Feet
dailysb22 = dailysb22 %>% mutate(waveheight = waveheight * 3.28084)
# Changing Wind Speed from meters per second into knots
dailysb22 = dailysb22 %>% mutate(windspeed = windspeed * 1.94384)
```

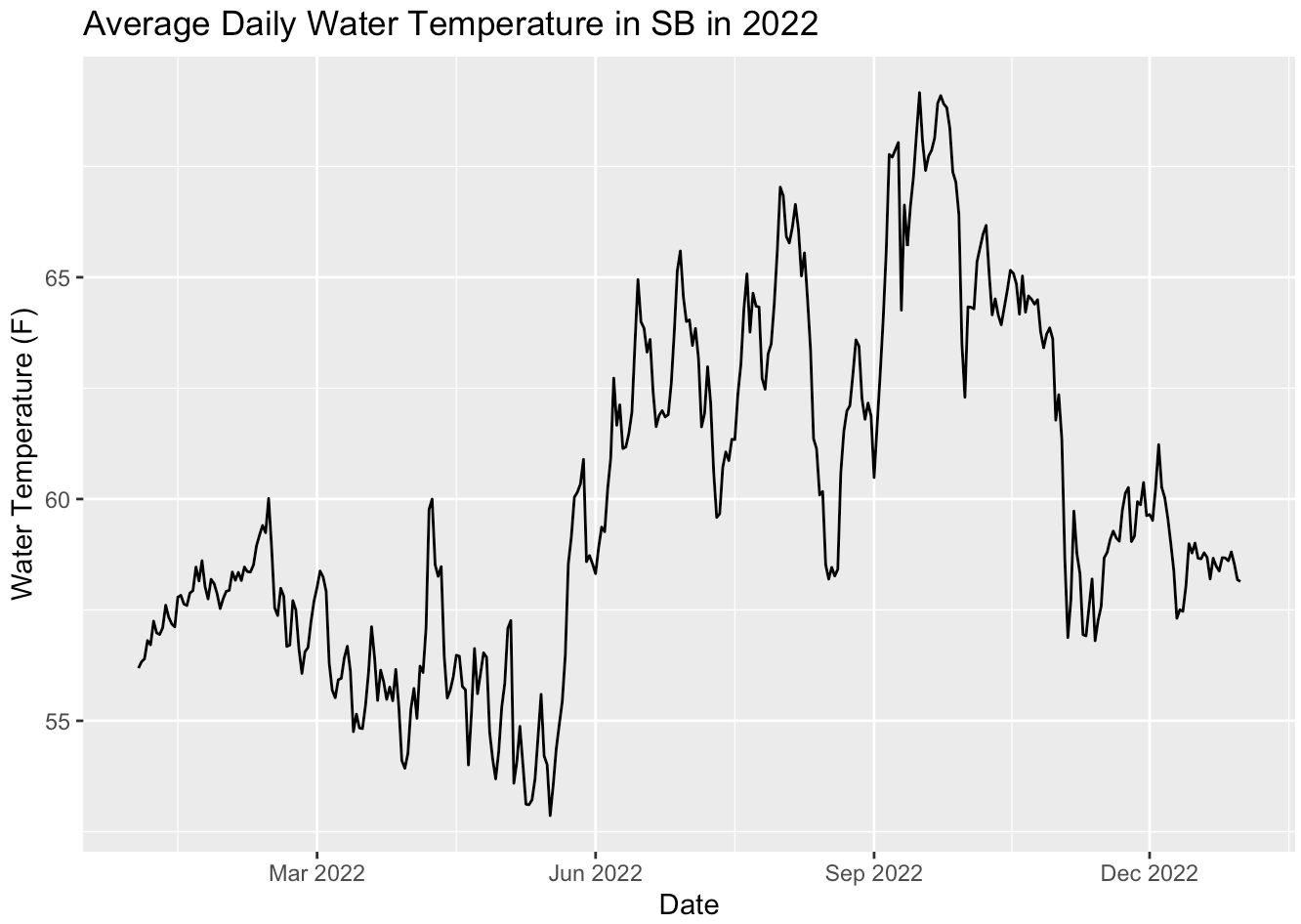
```
##       date               airtemp        watertemp     wavedirection
##  Min.   :2022-01-01   Min.   :49.77   Min.   :52.86   Min.   :176.8
##  1st Qu.:2022-04-02   1st Qu.:56.65   1st Qu.:57.12   1st Qu.:261.8
##  Median :2022-07-02   Median :58.53   Median :58.99   Median :265.8
##  Mean   :2022-07-02   Mean   :59.15   Mean   :60.01   Mean   :264.4
##  3rd Qu.:2022-10-01   3rd Qu.:61.59   3rd Qu.:63.31   3rd Qu.:268.8
##  Max.   :2022-12-31   Max.   :69.84   Max.   :69.16   Max.   :279.2
##    waveperiod       waveheight      windspeed      winddirection
##  Min.   : 4.278   Min.   :1.180   Min.   : 2.187   Min.   :105.5
##  1st Qu.: 7.528   1st Qu.:2.920   1st Qu.: 5.329   1st Qu.:208.0
##  Median : 9.837   Median :3.718   Median : 7.808   Median :238.4
##  Mean   :10.029   Mean   :3.906   Mean   : 8.296   Mean   :230.5
##  3rd Qu.:12.413   3rd Qu.:4.759   3rd Qu.:10.124   3rd Qu.:256.7
##  Max.   :18.348   Max.   :9.082   Max.   :21.830   Max.   :302.2
```

Our data set is now in great shape! We can now proceed and begin our analysis!

Visual EDA

Before looking at the correlation heat map of our data set, it is important to have an idea of how water temperature fluctuates throughout the year. This may provide valuable insight for interpreting relationships and identifying trends in our data for use later in our analysis.

```
p_waterTemp_year <- dailysb22 %>% ggplot(aes(y=watertemp, x=date)) +
  geom_line() +
  scale_x_date(date_labels = "%b %Y", date_breaks = "3 month") +
  labs(x = "Date", y="Water Temperature (F)") +
  ggtitle(label="Average Daily Water Temperature in SB in 2022")
p_waterTemp_year
```



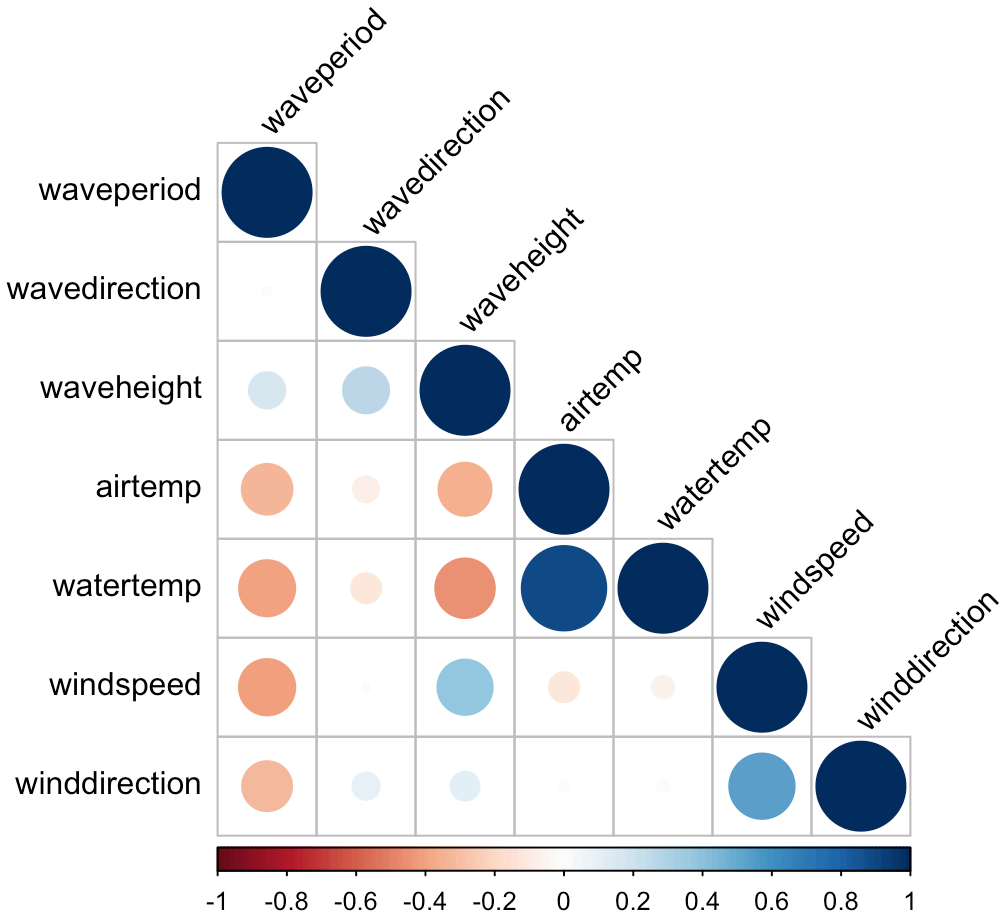
As expected, water temperature demonstrates a clear seasonal pattern. We can expect such seasonality to play a significant role in drawing conclusions from our visual analysis.

Correlation Map

Now, let's consider the correlation heat map of all the numeric variables. This visualization can help us understand hidden relationships between variables and identify strong correlations for use later in building our models.

```
# choosing numeric variables from the data set
num_dailysb22 <- dailysb22 %>% select_if(is.numeric)

# printing correlation heat map
corrplot(cor(num_dailysb22), type = "lower", order = "hclust",
     tl.col = "black", tl.srt = 45)
```



Our correlation heat map reveals some interesting patterns among the variables.

First, we observe a strong inverse relationship between wave period and water temperature. This suggests that longer wave periods are expected more during winter, aligning with the expectation that longer periods are correlated with lower water temperatures.

Additionally, there is a strong positive correlation between wave height and wind speed. This correlation is likely due to the fact that wind generates swell; thus, increased wind speed correlates with higher wave heights.

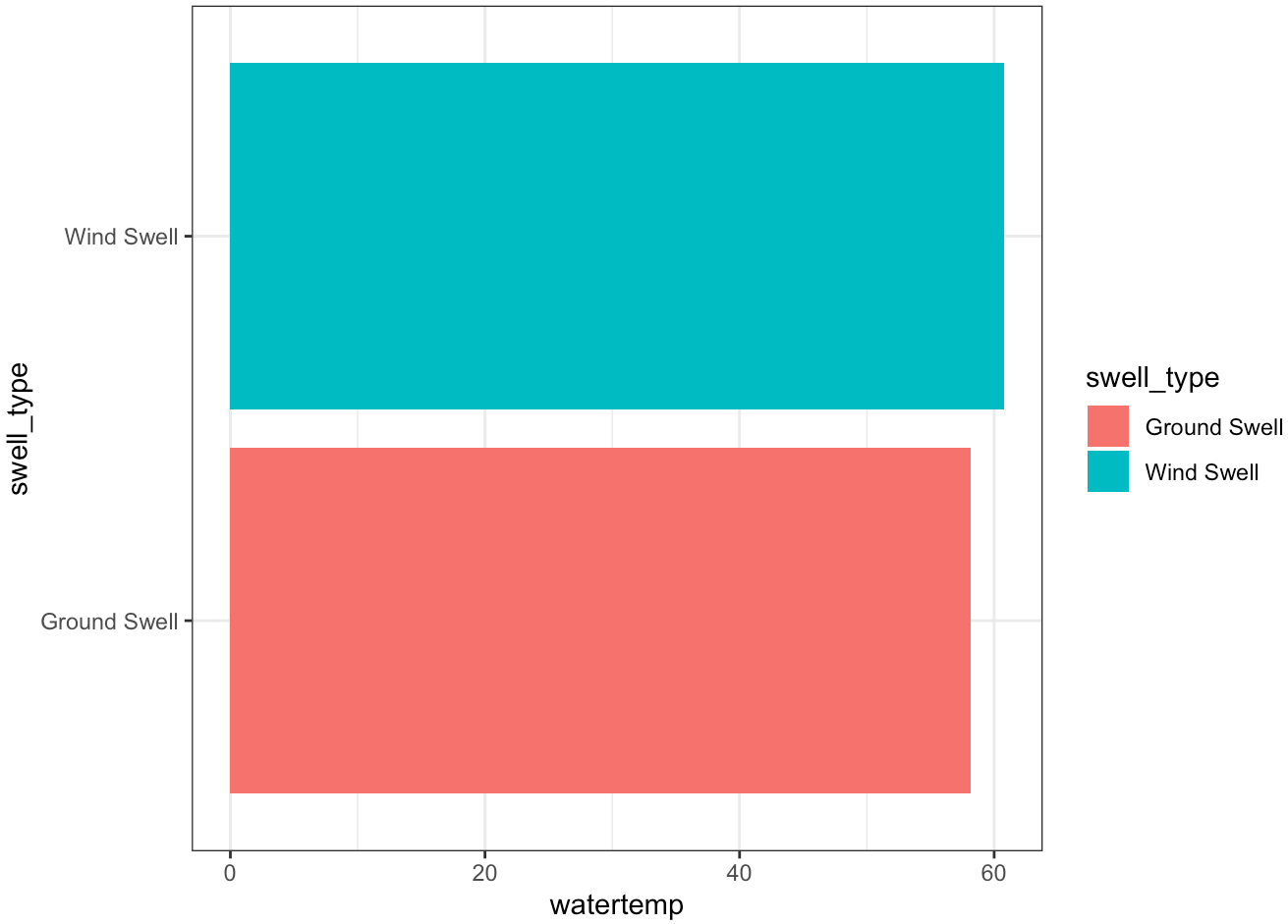
Plots of Interesting Relationships

Let's compare water temperature against the different types of swell: wind swell and ground swell. A wind swell is a swell with a period shorter than 12 seconds, and a ground swell has a period of 13 seconds or greater. To visualize this, let's make a factor variable for each type of swell and plot them against the water temperature.

Here I will make and visualize such variables.

```
# creating factor encoded swell variables
dailysb22$swell_type <- as.factor(ifelse(dailysb22$waveperiod <= 12, "Wind Swell",
                                         "Ground Swell"))
```

```
# creating bar plot to visualize average water temp and swell type
p_swellType_WTemp <- dailysb22 %>% ggplot(aes(y = swell_type, fill = swell_type,
                          x = watertemp)) +
  stat_summary(fun = "mean", geom = "bar",
               # calc and display mean water temp, position bars side by side
               position = "dodge") + theme_bw()
p_swellType_WTemp # display
```



Our analysis indicates that mean water temperature is generally higher during a wind swell than a ground swell. However, to determine if this pattern holds across different times of the year, we will plot water temperature against swell type for each season.

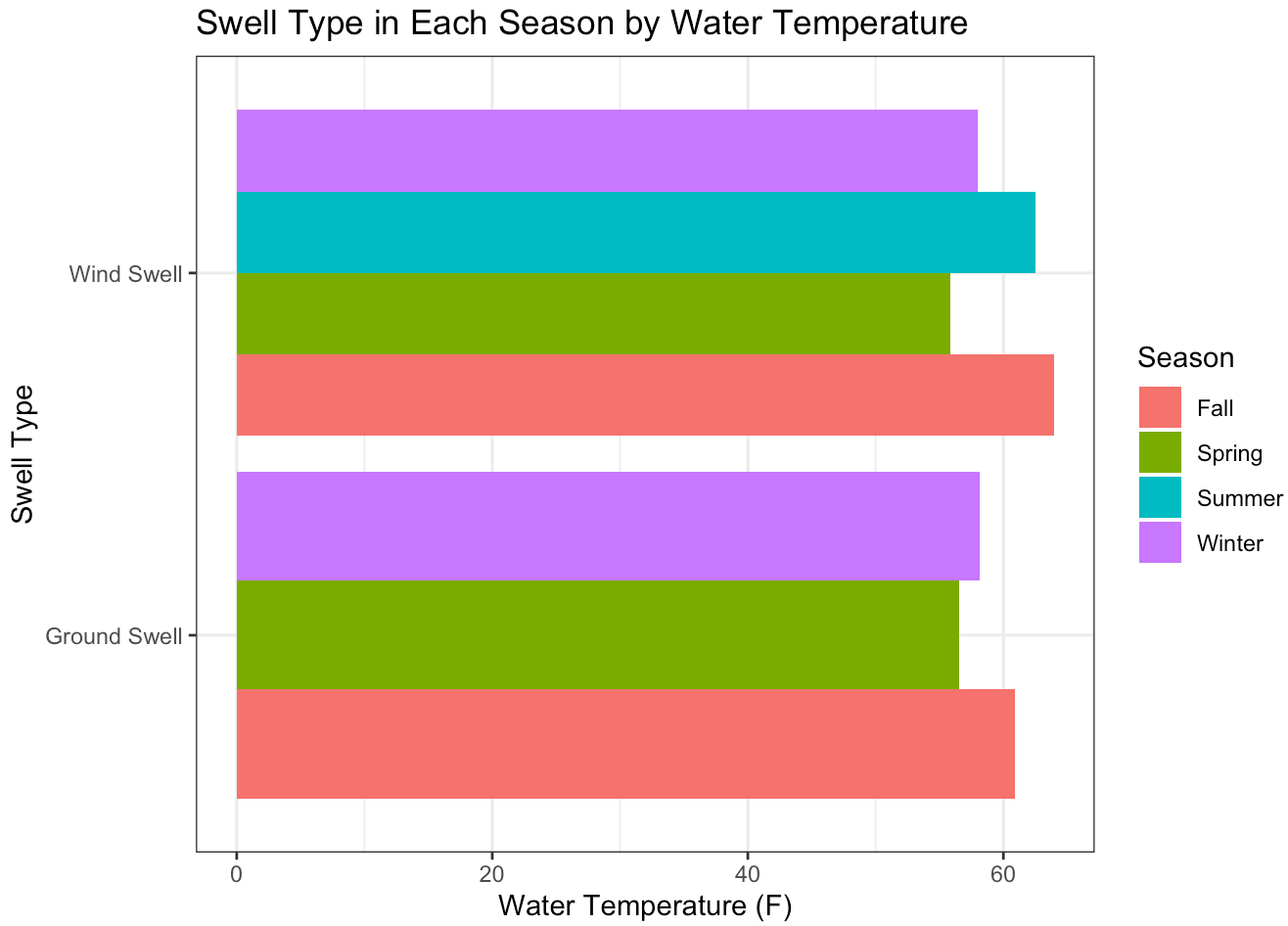
We need to make a new variable for the season. I will use the date information to classify observations by their respective seasons.

```
# creating factor variable for each season
dailysb22$Season <- as.factor(
                    ifelse(month(dailysb22$date) %in% c(3, 4, 5), "Spring",
                    ifelse(month(dailysb22$date) %in% c(6, 7, 8), "Summer",
                    ifelse(month(dailysb22$date) %in% c(9, 10, 11), "Fall", "Winter")))
)
```

Here is a bar plot of water temperature against swell type by season.

```
# baar plot to visualize water temp by swell type and season
p_swellType_Season_Temp <- dailysb22 %>%
  ggplot(aes(y = swell_type,fill = Season, x = watertemp)) +
  stat_summary(fun = "mean", geom = "bar",
               # position avg water temp bars side by side
               position = "dodge") +
  labs(x = "Water Temperature (F)", y="Swell Type") +
  ggtitle(label="Swell Type in Each Season by Water Temperature") + theme_bw()

p_swellType_Season_Temp # display
```



Not only do we observe that, on average, the water temperature is higher during shorter period swells compared to longer period swells, but our assumption about ground swells being more prominent in colder parts of the year is supported by the data.

With a solid understanding of our data, we are now ready to proceed with building and evaluating models.

Building Models

Splitting Data into Training and Testing Sets

The first step in building our models is to split the data into training and testing sets.

To mitigate the risk of over fitting, I will exclude the date column from our data set. The information related to date and water temperature is now effectively captured in other variables we have created.

To create our training and testing sets, I will stratify based on the water temperature variable. This ensures that both sets maintain an appropriate distribution of water temperature relative to one another, providing optimally balanced data for model training and evaluation.

```
# create initial split of data set, 70% in training 30% in testing set
wtmp_split <- initial_split(dailysb22, prop = 0.70,
                            strata = "watertemp")

wtmp_train <- training(wtmp_split) # extract training set from split
wtmp_test <- testing(wtmp_split) # extract testing set from split
```

Let's verify that we have the appropriate amount of observations in both the training and testing sets. Ensuring that these sets are balanced and sufficiently representative is essential for the robustness and accuracy of our model.

```
dim(wtmp_train) # checking dimension of training set
```

```
## [1] 253   7
```

```
dim(wtmp_test) # checking dimension of testing set
```

```
## [1] 112   7
```

We are good to go!

In our model evaluation process, we will use k-fold cross validation to compare model performance. A 5-fold cross validation will allow us to assess model performance by evaluating them on multiple subsets of data.

```
wtmp_folds <- vfold_cv(wtmp_train, v = 5, strata = watertemp)
```

Making an Initial Recipe

When building our models, we must first make a recipe specifying the relationship between our predictors and our response variable, water temperature, using our training data set. This recipe serves as the fundamental structure for how the predictors should relate to our target variable.

```
# specify recipe for preprocessing data to be used in making models
wtmp_recipe <- recipe(watertemp ~ airtemp + watertemp + wavedirection +
                                  waveperiod + waveheight + windspeed +
                                    winddirection, data=wtmp_train) %>%
  # Normalize all predictor variables to have
  # a mean of 0 and standard deviation of 1
                     step_normalize(all_predictors())
```

Fitting the Models

We will be develop a Linear Regression model, a kNN (k Nearest Neighbors) model, a Polynomial Regression model, and a Random Forest Model. Then, we will compare the accuracy of each model to determine which is best.

To build a model, we first have to specify a model type we want then an appropriate workflow. In our workflows, we will tune hyper parameters (which vary by model) in order to optimize their performance. Since the response variable, water temperature, is quantitative, each model will be a regression model.

Once the workflows and tuning grids for each model have been designed, we will run the models in separate scripts and import the results back into this analysis. This method will be more efficient than running all models in this analysis.

Linear Regression

First, we will make the model and workflow for our linear regression model, which does not involve any hyper parameters to tune.

```
# model for linear regression
model_lin_reg <- linear_reg() %>% set_engine("lm")
```

```
wf_lin_reg <- workflow() %>%
            add_model(model_lin_reg) %>%
            add_recipe(wtmp_recipe) # workflow for linear regression
```

Now, I will save the linear regression workflow to run it in another .R, ensuring we can proceed efficiently with the analysis.

k-Nearest Neighbors (kNN)

In this section, we will tune the neighbors hyper parameter to optimize our kNN model. This will help determine the ideal number of neighbors for the best model possible.

```
# defining k-Nearest neighbors model
model_knn <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn") # model
```

```
# creating k-Nearest neighbors work flow
wf_knn <- workflow() %>%
            add_model(model_knn) %>%
            add_recipe(wtmp_recipe)
```

```
# creating tuning grid on neighbors hyper parameter
grid_knn <- grid_regular(neighbors(range = c(1,10)),
                         levels = 5)
```

Random Forest

In our Random Forest model, we will tune three hyper parameters: mtry, trees, and min_n.

mtry: This represents the number of predictors that may be randomly sampled in each split in our random forest. trees: This is the total number of trees contained in the model. min_n: This is the minimum amount of data points required at each node before splitting further.

```
#  defining random forest model
model_rand_for <- rand_forest(mtry=tune(),
                              trees=tune(),
                              min_n = tune()) %>%
                      set_mode("regression") %>%
                      set_engine("ranger", importance = "impurity")
```

```
# defining random forest workflow
wf_rand_for <- workflow() %>%
            add_model(model_rand_for) %>%
            add_recipe(wtmp_recipe)
```

Now, I will create a grid of possible values for tuning each hyper parameter in the Random Forest model.

```
# tuning grid for random forest hyper parameters: mtry, trees, min_n
grid_random_for <- grid_regular(mtry(range = c(1, 6)),
                                trees(range = c(200,600)),
                                min_n(range = c(3,15)),
                                levels = 7)
```

Modified Polynomial Recipe

For our Polynomial Regression model, we will slightly modify the recipe to allow for polynomial expansion of our predictors at higher degrees. We will tune the degree hyper parameter and include the following predictors for polynomial expansion: - Air Temperature - Wave Period - Wave Height - Wind Speed - Wind Direction

Note that Wave Direction will not be selected for polynomial expansion. I am exclusively choosing predictors which are most easily seen with the naked eye, as such predictors more closely align with the context of our analysis.

```
# modified polynomial regression recipe
rec_poly <- wtmp_recipe %>% step_poly(airtemp,
                                      waveperiod,
                                      waveheight,
                                      windspeed,
                                      winddirection,
                                      degree = tune())
```

Polynomial Regression

In this section, we will define the Polynomial Regression model and workflow and tune the degree hyper parameter.

```
# defining polynomial regression model
poly_mod <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")
```

```
# creating polynomial regression workflow
wf_poly <- workflow() %>%
  add_model(poly_mod) %>%
  add_recipe(rec_poly)
```

```
# creating tuning grid for degree hyper parameter
grid_poly <- grid_regular(degree(range = c(1,10)), levels = 5)
```

Tuning the Models

Now that we have everything we need- four model workflows and three tuning grids- we can proceed with tuning the models according to our specifications.

```
# there is no linear regression hyper parameter to tune

# tuning knn model hyper parameter
knn_tune <- tune_grid(
  wf_knn,
  resamples = wtmp_folds,
  grid = grid_knn)

# tuning random forest hyper parameters
random_forest_tune <- tune_grid(
  wf_rand_for,
  resamples = wtmp_folds,
  grid = grid_random_for)

tune_poly <- tune_grid(wf_poly, # tuning polynomial regression hyper parameter
                       resamples = wtmp_folds,
                       grid = grid_poly)
```

Now, we can load our results to compare which models had lower Root Mean Square Error (RMSE). Once we have identified the best-performing models, we will proceed to fit the testing set to them in order to further assess their performance and effectiveness.

Storing RMSE of Each

Linear Regression

```
set.seed(1112)
# Fit linear regression model using resampling and store the results
lreg_fit <- fit_resamples(wf_lin_reg, resamples = wtmp_folds)

# Collect RMSE metrics from fitted linear regression model
# select the first row of RMSE metrics
rmse_lreg <- collect_metrics(lreg_fit) %>% slice(1)
```

Random Forest

```
set.seed(1112)
# Retrieve the best RMSE metric from the tuned Random Forest model
rmse_rand_forest <- show_best(random_forest_tune, metric = 'rmse', n=1)
# Display the best RMSE for the Random Forest model
rmse_rand_forest
```

```
## # A tibble: 1 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4   400     3 rmse    standard    1.40     5  0.0861 Preprocessor1_Model0…
```

k-Nearest Neighbors

```
set.seed(1112)
# Retrieve the best RMSE metric from the tuned kNN model
rmse_knn <- show_best(knn_tune, metric = 'rmse', n=1)
# Display the best RMSE for the kNN model
rmse_knn
```

```
## # A tibble: 1 × 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        10 rmse    standard    1.63     5   0.102 Preprocessor1_Model5
```

Polynomial Regression

```
set.seed(1112)
# Retrieve best RMSE metric from tuned Polynomial Regression model
rmse_poly <- show_best(tune_poly, metric = 'rmse', n=1)
# Display the best RMSE for the Polynomial Regression model
rmse_poly
```

```
## # A tibble: 1 × 7
##   degree .metric .estimator  mean     n std_err .config
##    <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   3.25 rmse    standard    1.54     5  0.0586 Preprocessor2_Model1
```

To ensure we retain these RMSE values, I will save the results and then load them back into our workspace for future reference.

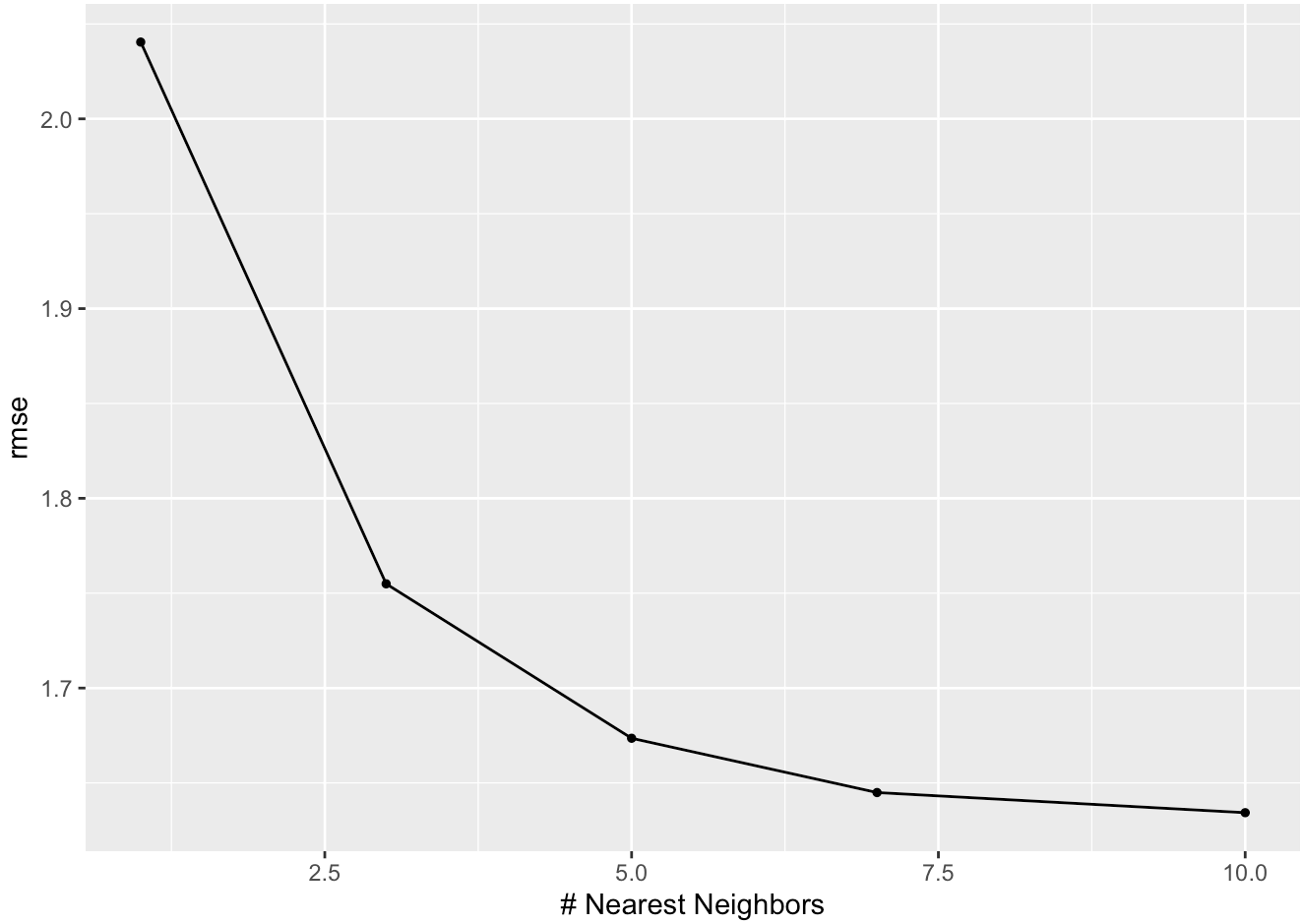
Model Comparisons

Now that we have each model's RMSE, we can compare which model or models achieved the best performance.

Autoplots

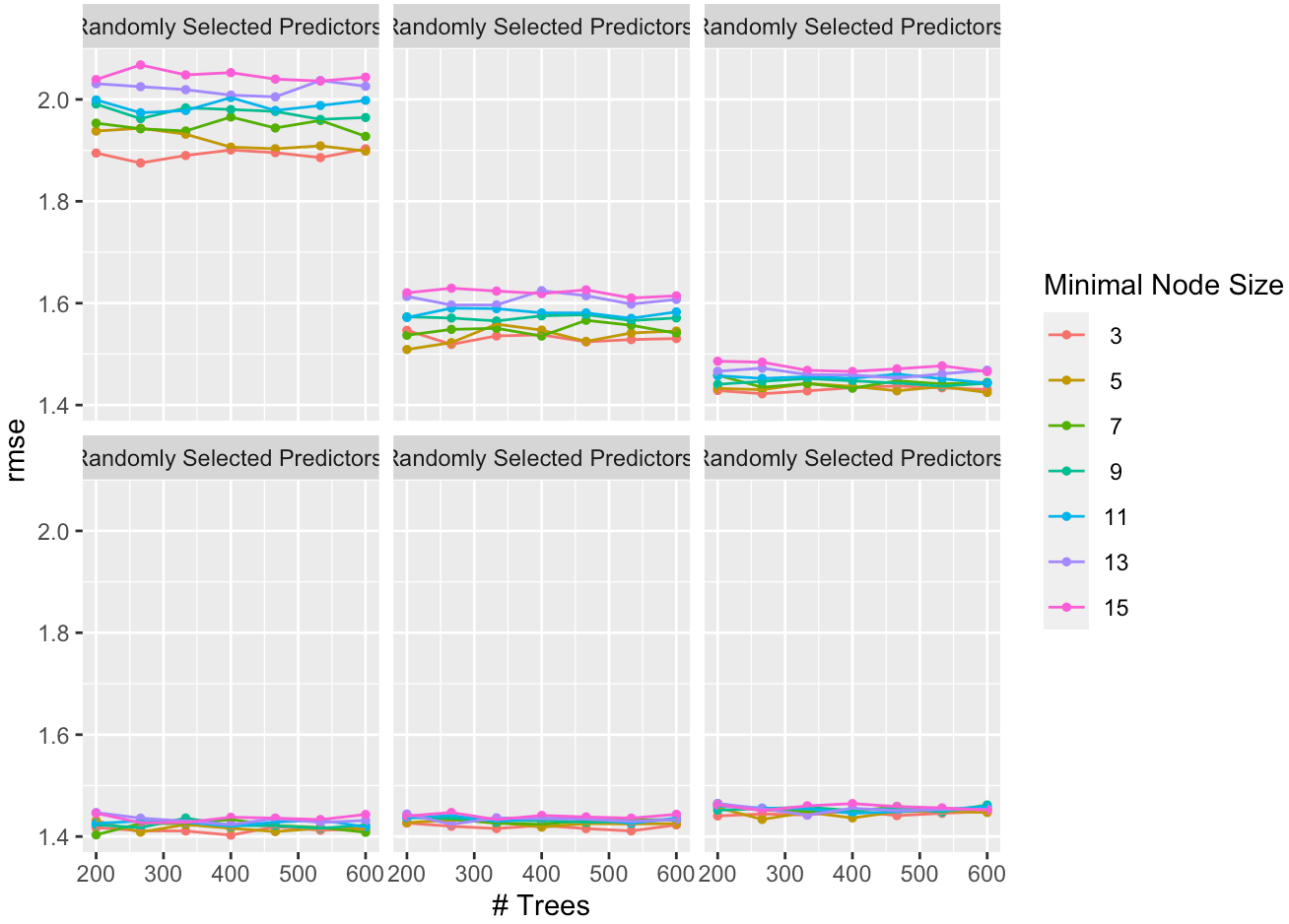
Before we conduct a thorough comparison of the models, it may be beneficial to visualize the tuning process for each one. To achieve this, we will examine the autoplots of our tuned models.

```
# autoplot of kNN RMSE
autoplot(knn_tune, metric = 'rmse')
```



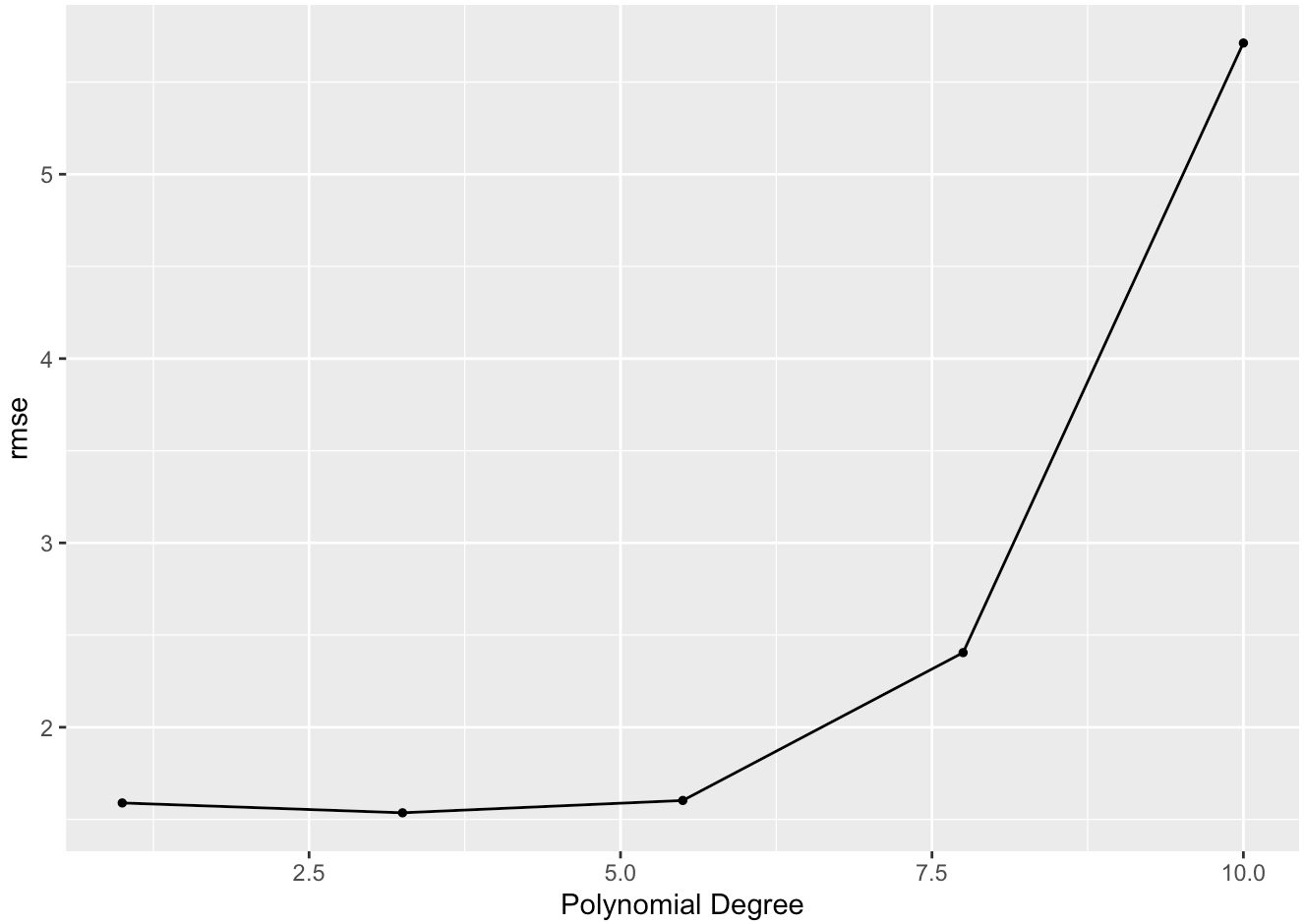
Upon reviewing the autoplot of the k-Nearest Neighbors model, it is clear that models with a higher number of neighbors performed better. The models with 7 to 10 neighbors demonstrate similar RMSE performance and are significantly more accurate than those models with fewer than 5 neighbors.

```
# autoplot of Random Forest RMSE
autoplot(random_forest_tune, metric = 'rmse') # random forest
```



In the autoplot of the Random Forest Model, it is noteworthy that models with smaller node sizes generally exhibited lower RMSE values compared to those with larger node sizes, particularly when the same quantity of predictors was selected. When node size is smaller, the model is able to create more splits which can better capture the nuanced relationships within the data. The observation that smaller node sizes generally produces smaller RMSE values suggests that such models were best suited to capturing the variability in the data, improving their predictive performance. Smaller node sizes, in this context, contribute to improved accuracy without over complicating the model.

```
# autoplot of polynomial regression RMSE
autoplot(tune_poly, metric='rmse')
```



In the autoplot of the Polynomial Regression, we observe model RMSE increases significantly beyond the 5th degree of the polynomial; lower order degrees exhibit less variable RMSE values, potentially indicating that higher degree polynomials may lead to over fitting. From this we can infer that while polynomial regression may capture more complex relationships in the data, model performance decreases beyond some threshold- in this case, the 5th degree.

Ranking Models

In this section, we will now rank the performance of our 4 models based on their RMSE values. Root Mean Square Error is an evaluative metric of the accuracy of regression models which measures the average magnitude of error between the predicted and observed values. The lower the RMSE, the better a model fits the data.

First, we will load in the stored RMSE results from the models we have built. By comparing the RMSEs directly we can create a descriptive overview of the models performances in relation to one another.

Once these RMSE scores have been loaded, they will be compiled into a tibble and arranged so we can easily identify the model exhibiting the lowest error. From here, we will deduce which model or models were most effective in predicting water temperature based on the features we have explored.

```
## [1] "rmse_lreg"        "rmse_knn"         "rmse_rand_forest" "rmse_poly"
```

```
set.seed(1112)

# Combine mean RMSE values from each model into a single vector RMSE
RMSE = c(rmse_lreg$mean,
         rmse_knn$mean,
         rmse_rand_forest$mean ,
         rmse_poly$mean)

#Create a character vector for the model names
Model <- c("Linear Regression",
           "k Nearest Neighbors",
           "Random Forest",
           "Polynomial Regression")

# Combine the model names and corresponding RMSE values into tibble
# arrange it by RMSE
compare <- tibble(Model, RMSE) %>% arrange(RMSE)
compare <- tibble(Model,RMSE) %>% arrange(RMSE)
# display ranked RMSE table
compare
```

```
## # A tibble: 4 × 2
##   Model                 RMSE
##   <chr>                <dbl>
## 1 Linear Regression    0.870
## 2 Random Forest        1.40
## 3 Polynomial Regression 1.54
## 4 k Nearest Neighbors  1.63
```

From the RMSE ranking, we deduce that the best performing model was the Linear Regression model, followed by the Random Forest, Polynomial Regression, then k-Nearest Neighbors. This suggests that the relationships between water temperature and its predictors are likely linear in nature. Meteorological data often exhibits trends that follow some sort of seasonality or generally linear relationships- such as in this data where we can see the linear connection between wave periods, time of year, and ultimately water temperature as we identified previously in the correlation analyses.

Not only does this outcome reinforce the validity of our approach, but it also implies that the more complex models we examined introduced some unnecessary complexity without improving accuracy.

Testing Our Best Model

Now that we have identified the linear regression model as the most appropriate for our data, we can proceed by fitting the model to our testing set. This will allow us to evaluate the model's performance on unseen data and conclusively assess its predictive accuracy.

To evaluate the final model's validity, we will use three metrics: RMSE, R-Squared (rsq/R^2), and Mean Absolute Error (MAE).

R^2 is a measure of how much variation in the target variable can be attributed to the predictors in the model. A value closer to 1 explains that a larger proportion of the variance in the outcome is attributable to the model's predictors. On the other hand, an R^2 value closer to 0 suggests that the predictors explain less of the variance,

implying that the variance is due to randomness or factors not captured by the model.

MAE quantifies how far off the predicted values are from the actual values by calculating the absolute differences between them. A lower Mean Absolute Error is indicative of a better model performance, as it explains that the predictions are closer to the actual values.

```
# define performance metrics: RMSE, rsq, MAE
multi_metric <- metric_set(rmse, rsq, mae)

# fit the linear regression model to training data set
wtmp_fit <- fit(wf_lin_reg, wtmp_train)

# generate predictions for the testing data using fitted model
# bind the predicted values with actual water temp data from testing set
wtmp_predict <- predict(wtmp_fit, wtmp_test) %>%
  bind_cols(wtmp_test %>% select(watertemp))
```

```
# evaluate performance of model using metrics constructed above
multi_metric(wtmp_predict, truth = watertemp, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.868
## 2 rsq     standard       0.844
## 3 mae     standard       0.684
```

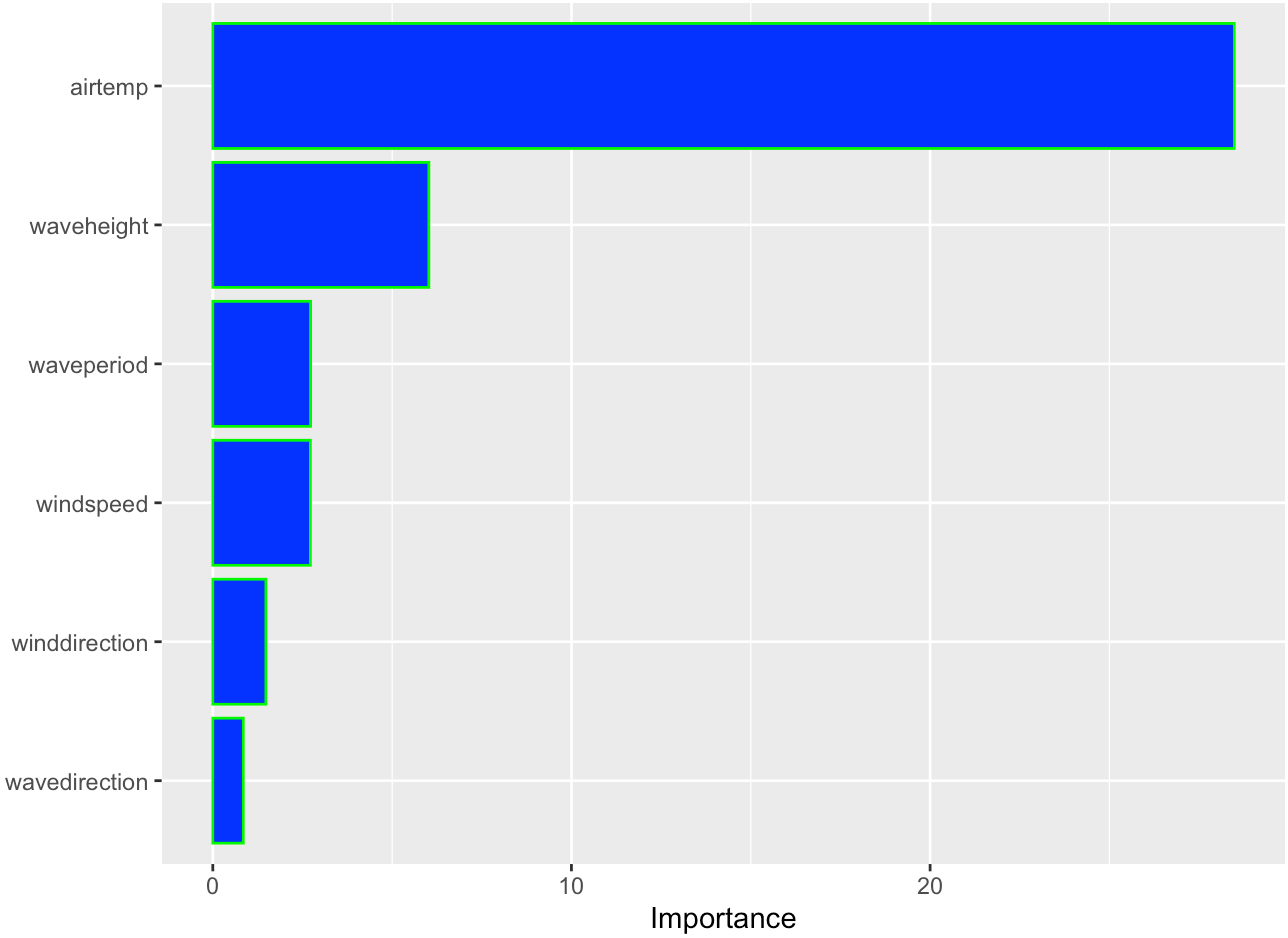
According to our metrics, the

$$R^2$$

of our set shows that 84.4% of our variation in the water temperature can be described by the combination of predictors: air temperature, wind direction, wave direction, wind speed, wave period, and wave height. This suggests the relationships between these variables and water temperature is predominantly linear, which further validates the appropriateness of our regression model for this data set.

Let's take a closer look at the significance of each predictor in our model. By examining the Variable Importance Plot (VIP), we can visualize the influence of each predictor variable in predicting water temperature. Predictors with a higher importance in the VIP plot were more influential than those with lesser importance.

```
# extract fitted model from workflow, engine used in fitting model
wtmp_fit %>% extract_fit_engine() %>% vip(aesthetics = list(fill = "blue", color = "gree
n")) # display VIP plot
```



According to the VIP plot, air temperature emerges as the most influential predictor of water temperature, followed by wave height. This aligns with our earlier assumptions that seasonal variations significantly influence water temperature. Air temperature being such a significant predictor of water temperature reinforces our understanding that broader environmental factors have a major impact on water conditions.

Conclusion

In my effort to create the most effective machine learning model for predicting ocean water temperature, I developed four models: Linear Regression, k-Nearest Neighbors, Polynomial Regression and Random Forest.

Ultimately, the model with the lowest RMSE (Root Mean Square Error) was the Linear Regression model, also achieving an R^2 of 84.4%, and a Mean Absolute Error of 0.684. This MAE indicates that on average, the model's predictions for water temperature were 0.684 degrees away from the actual measured water temperature. This suggests that the ocean's water temperature can be reliably modeled in a linear combination of observable factors such as air temperature, wave height, wind direction, wind speed, and wave period.

I'm pleased that the Linear Regression model was so effective in predicting water temperature, as it is easily implemented and aligns with the linear relationships we speculated on earlier in our analysis.

So, what does this mean? Well, if it's cold outside, the ocean is probably cold too.

In a secondary analysis, I would like to see how the model's performance changes when excluding variables indicative of seasonal trends. Specifically, I'm interested in building a model that does not incorporate broader weather phenomena such as air temperature and time of year. This would allow me to focus on ocean-specific

factors like wave height, wave period, wind direction, and wind speed to determine how these alone predict water temperature, without the influence of seasonal patterns. It would be insightful to see how much predictive power such oceanic conditions hold on their own, without the dominant effects of seasonal variations.

Sources

The data was collected from NOAA's NDBC Station 46053 LLNR 196 Buoy (https://www.ndbc.noaa.gov/download_data.php?filename=46053h2022.txt.gz&dir=data/historical/stdmet/).

All of the background knowledge I have provided has been from years spent learning about the ocean from first-hand experience.