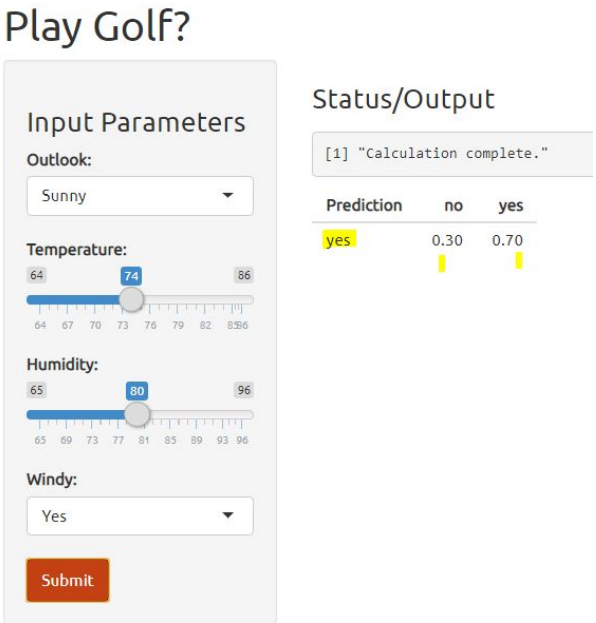
Building a ML / Data-Driven Web Application in R

- **Again using the Shiny web framework**

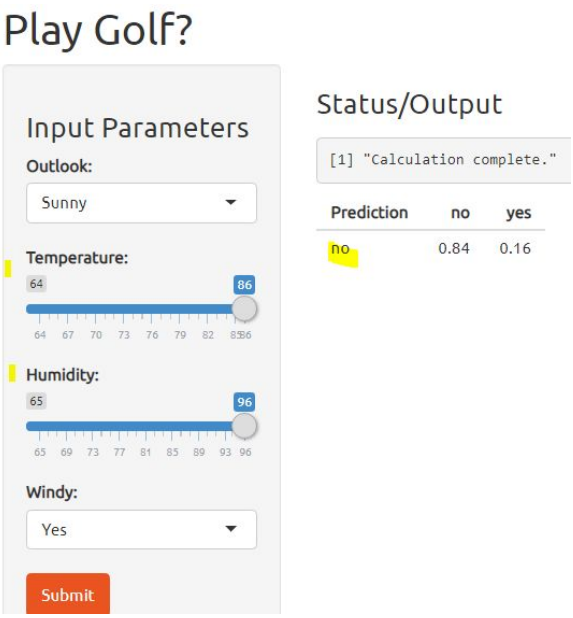
- This will make use of the **random forest algorithm**. It aims to predict whether or not to play golf as a function of the **input weather parameters**.
- Recall def'n of **parameter** in stats terms: sigma or p; it's a value that tells you something about a population ((weather outlook, temperature, humidity and wind)
    - Difference between a stat and a parameter?
        - Both are similar since they are both descriptions of groups, however, stats describe a **sample whereas a parameter describes the WHOLE pop'n**
- This web app will be based on the Weather dataset from weka data mining software
- Link: https://github.com/dataprofessor/data/blob/master/weather-weka.csv

- Note: install libraries you don't have; i'm missing RCurl and randomForest, hence the code below
- **install.packages(c("randomForest", "RCurl"))**

- Looking at the data, the 4 variables are outlook, temperature, humidity, windy
    - The **class label is play** (whether or not to play golf based on variables)
        - This is a function of the weather conditions; i.e. whether it is sunny, temp, humidity is high or low or medium, whether it is windy; true or false etc.



- - with all variable factors being taken into account, no has a 30% probability while yes has 70% probability
    - Hence, the overall decision / prediction being stated as yes

- 

- with hot and humid temperatures, the decision would obviously be no

What does the data set actually look like?

- **view(weather)**
- 

	outlook	temperature	humidity	windy	play
1	sunny	85	85	FALSE	no
2	sunny	80	90	TRUE	no
3	overcast	83	86	FALSE	yes
4	rainy	70	96	FALSE	yes
5	rainy	68	80	FALSE	yes
6	rainy	65	70	TRUE	no
7	overcast	64	65	TRUE	yes
8	sunny	72	95	FALSE	no
9	sunny	69	70	FALSE	yes
10	rainy	75	80	FALSE	yes
11	sunny	75	70	TRUE	yes
12	overcast	72	90	TRUE	yes
13	overcast	81	75	FALSE	yes
14	rainy	71	91	TRUE	no

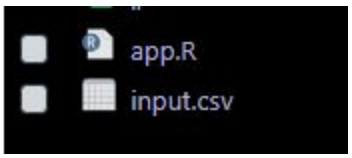
  - 5 cols: 4 variables/ parameters, and 1 class label
- Quicker view of some data, first 6 rows; use: **head(weather)**
- To determine the data type of the data set / to extract the data type of the data set?
  - **str(weather)**
  -
```
> str(weather)
'data.frame':   14 obs. of  5 variables:
 $ outlook    : Factor w/ 3 levels "overcast","rainy",..: 3 3 1 2 2 2 1 3 3 2 ...
 $ temperature: int  85 80 83 70 68 65 64 72 69 75 ...
 $ humidity   : int  85 90 86 96 80 70 65 95 70 80 ...
 $ windy      : logi  FALSE TRUE FALSE FALSE FALSE TRUE ...
 $ play       : Factor w/ 2 levels "no","yes": 1 1 2 2 2 1 2 1 2 2 ...
```

- “Play” is a categorical label

Random Forest Model - Will study this more...

- A randomForest model will be created using the four variables comprising of outlook, temp, humidity, windy as the input variable
  -
```
# Build mordel
model <- randomForest(play ~ ., data = weather, ntree = 500, mtry = 4, importance = TRUE)
```
    - The “play” variable will be used as the output variable OR IOW the variable we want to predict
    - Data is “weather”
    - Number of trees: 500
    - Since there are 4 input variables, will use “mtry = 4”

- Will learn more about the Random Forest Model itself; pretty complex...

- Will apply model to make a prediction
  - 

- notice the input file; the model will be applied to this, to make a prediction
  -
```
121 Output <- data.frame(Prediction=predict(model, test), round(predict(model, test, type="prob"), 3))
122 print(Output)
123
```

```
    # Status/Output Text Box
    output$contents <- renderPrint({
        if (input$submitbutton > 0) {
            isolate("Calculation complete.")
        } else {
            return("Server is ready for calculation.")
        }
    })

    # finally, a prediction will be made using the model generated earlier
    #by means of the random forest algorithm
    #and apply the prediction model to predict the input values from the user


    # Prediction results table
    output$tabledata <- renderTable({
        if (input$submitbutton > 0) {
            isolate(datasetInput())
        }
    })
    #and once the prediction is made, it will be sent here into
    #"output$tabledata" as the function dataset input
    # then it is going to render the table as we will see in the web app

}
```

- The table being rendered / the output table

-

```
    output$contents <- renderPrint({
        if (input$submitbutton > 0) {
            isolate("Calculation complete.")
        } else {
            return("Server is ready for calculation.")
        }
    })

    # finally, a prediction will be made using the model generated earlier
    #by means of the random forest algorithm
    #and apply the prediction model to predict the input values from the user


    # Prediction results table
    output$tabledata <- renderTable({
        if (input$submitbutton > 0) {
            isolate(datasetInput())
        }
    })
    #and once the prediction is made, it will be sent here into
    #"output$tabledata" as the function dataset input
    # then it is going to render the table as we will see in the web app

}   # NOTE:  there are 2 outputs being generated:
    # 1. output$content
    # 2. output$tabledata
#so, these two outputs will be sent to the UI component "mainPanel" (around line 93)
```

```
    mainPanel(
        tags$label(h3('Status/Output')), # Status/Output Text Box
        verbatimTextOutput('contents'),
        tableOutput('tabledata') # Prediction / results table
    )
)
```