Project 5: Predictive Modeling (A.K.A Introduction to Machine Learning)

In this project you will learn about how to use R to make predictions based on data analysis. S

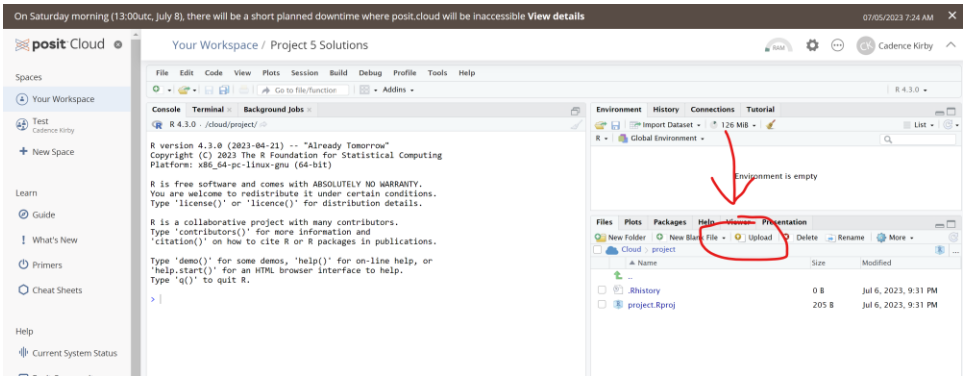
General:

You will work through the project at your own pace. The project will often ask you to do things that you do not know how to do. Do not worry, this is intentional. Any time you feel stuck, ask a TA or the instructor for help. Alternatively, use Google/YouTube/Canvas to find resources to help you.

1. You have been given an incredible opportunity! A genie has granted you the ability to travel back in time to April of 1912 to take a ride on the *Titanic*. Unfortunately, to avoid irreparably altering the future (and potentially erasing your existence from space-time), you cannot warn the captain about the iceberg that is sure to sink the ship. Luckily, the genie is allowing you to choose your attributes for this journey. You can choose your age, sex, passenger class, ticket price and boarding port. Using a dataset of *Titanic* passengers, your job is to use predictive modeling to determine what attributes you should give yourself in order to maximize the chances of your survival.

2. Open Posit Cloud and create a new project in the same workspace that you completed Project 4. Name your project "Project 5".

3. Download the "titanic.csv" dataset. It can be found in Canvas -> Files -> Projects -> Project 5. Import this dataset into your Posit Cloud project by clicking Upload -> Choose File -> "titanic.csv".



4. In the console, type install.packages("ggplot2") and run that command.

5. Create an R script by clicking CTRL+ALT+SHIFT+N. Alternatively you can go to the top of your screen and click File -> New File -> R Script. Name this script "main.r". At the top of the script, type *library(ggplot2)*

6. Import the titanic dataset using the read.csv function, just like you imported the climate dataset in Project 3. Use the print() and head() functions to display the first few rows of the data. Here is some information about the unintuitive columns of your data:

Survived: =1 if passenger survived, =0 if passenger did not survive

Pclass: The passenger class of a passenger. 1 = first class, 2 = second class, 3 = third class

SibSp: The number of siblings and spouses a passenger had with them on the boat

Parch: The number of parents and children a passenger had with them on the boat

Ticket: The ID of a passenger's ticket

Fare: How much each passenger paid for a ticket

Cabin: The cabin a passenger was staying in (this column has many missing values)

Embarked: The port at which a passenger boarded the boat. C = Cherbourg, Q = Queenstown,

S = Southampton

You may notice that all the attributes you get to choose for your time-travelling "avatar" are attributes in the dataset.

7. Before we begin, let's get our terminology straight. Watch this video giving an introduction to predictive modeling. According to the video, what is the most common field people associate predictive modeling with? Respond as a comment in main.r.

8. Let's warm up by using the most basic machine learning model – linear regression. Watch this video to get an idea on what linear regression is and when we can use it. What type of data do we need on both our x-axis and y-axis to do linear regression? What is a residual? How are residuals related to a linear regression line? Respond to these questions as a comment in main.r.

9. You will now build a linear regression model in R. We are going to step away from the titanic dataset for this first model and use the iris dataset instead. This is because the iris dataset gives a very clean linear regression model, and the first predictive model you build in R should be clean! In general, when we build a predictive model in R we will follow the same three steps.

First, we divide the data into training data and testing data. Standard practice is to use 80% of our data to train our model, and the remaining 20% to test our model. *Why do you think it is important to save some of our data for testing? (Respond as a comment in main.r).*

Second, we use the training data to build our model.

Finally, we test the accuracy of our model using the testing data.

10. We are going to build a linear regression model in R that predicts the length of a petal from an iris flower based on the width of that petal. First, plot petal length versus petal width using the simple plot() function. Does it look like there is some kind of linear relationship between petal length and petal width? Respond as a comment in main.r.

11. There seems to be a linear relationship between petal length and petal width. Now let's find out quantitatively how strong this linear relationship is. Run this line of pseudocode:

```
# Psuedocode: Replace anything between asterisks **
# Output some statistics about the strength of the linear relationship between
# Petal.Length and Petal.Width
print(cor.test(iris$*attribute we want to predict*,
               iris$*attribute we want to use to make our prediction*))
```

This outputs a few statistics. Notably, we want to look at the values associated with "p-value" and "cor". "cor" represents a percentage correlation, so a "cor" value closer to 1.0 means there is more correlation between our attributes. You should already be familiar with p-values from Project 4. Do these two values (p-value and cor) indicate that there is a strong linear relationship between petal size and petal width? Respond as a comment in main.r.

12. Because there is a strong linear relationship between petal width and petal length, linear regression is an ideal predictive model for this scenario. Use this pseudocode to randomly split the iris data into test data and training data:

```
# Pseudocode: replace everything within backslashes \\
# Split iris dataset into 80% training data and 20% testing data
index = sample(1:nrow(\\dataset\\), round(0.8*nrow(\\dataset\\), 0), replace = FALSE)
train = \\dataset\\[index, ]
test = \\dataset\\[-index, ]
```

Why do we need to use the round function in this code? Respond as a comment in main.r.

13. Use this pseudocode to construct the linear regression model:

```
#Pseudocode: replace everything inside asterisks **
linRegModel = lm(*Attribute we want to predict* ~ *Attribute we use to make prediction*,
                 *data we want to use to train our model*)
print(summary(linRegModel))
```

From the Khan Academy video on linear regression (in addition to your knowledge of statistics from previous projects), you should know how to interpret the residuals information in the summary of our model. What is the residuals information telling us? Respond as a comment in main.r.

Next look at the adjusted R-squared value. This tells us what proportion of variation in petal length can be explained by variation in petal width.

Finally, write the equation of the line generated by our linear regression model. Hint 1: remember the equation of a line is y = mx+b where m is slope and b is y-intercept. Hint 2: print(linRegModel) to see coefficients to plug into line equation. Respond as a comment in main.r.

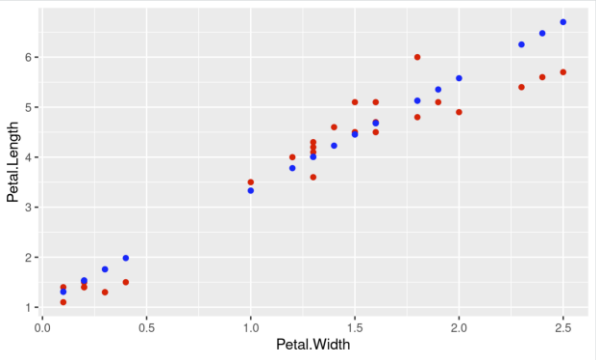
14. Plot the model by using the line plot(linRegModel). Hit enter to cycle through each plot. For each plot that shows up, do some research on what that plot is showing. For each plot, try to determine what you should be looking for to assess the robustness of the model. Record your results as a comment in main.r.

15. Let's now test our model using this test data. First, use your model to predict petal length based on the petal widths of the test data. Then, add these predicted lengths to the test data to compare them to the actual petal lengths. Here is some pseudocode:

```
#Pseudocode: replace everything inside asterisks **
prediction <- predict(*model*, *data we want to test with*)
test$prediction = *predicted values*
print(test)
```

Look at your predictions for petal length and compare them to the actual values for petal length. Are they similar? Respond as a comment in main.r.

16. Let's represent the accuracy of our model both visually and numerically. First, make a simple scatter plot of petal length versus petal width. Represent the actual values as red and the predicted values of blue. Your graph should look like this:



Next, we are going to calculate the mean squared error (MSE) of our regression line. Watch this video on squared error of linear regression lines. The mean squared error is simply the mean of all of the squared errors of the data points. As a comment in main.r, give two reasons why you think we square the error values. Then, use this pseudocode to the MSE of our entire model:

```
# Pseudocode to calculate MSE, replace everything in asterisks **

# Calculate squared differences
squared_diff <- (*column of actual values* - *column of predicted values*)^2

# Calculate mean squared error (MSE)
mse <- mean(squared_diff)
print(mse)
```

STOP: Check in with a TA or instructor before continuing

17. Let's go back to the *Titanic* dataset. We are going to make a linear regression model using multiple features (feature is just data science lingo for a predictor variable). When we use multiple features in a linear regression model, this is called multiple linear regression. Watch this video about multiple linear regression in R. Do you think multiple linear regression models can be made with categorical features? Explain why or why not as a comment in main.r.

18. Build a multiple linear regression model predicting survival. Use the features age, sex, passenger class, ticket price and boarding port, because these are the features we can customize in our time-travelling avatar. You will use the same steps that you did when building your basic linear regression model for the iris dataset: first split the data into 80% training data and 20% testing data (refer to step 12 if you forgot how to do this). Then, make your model like this:

```
# Build multiple linear regression model using five features
multiplelrm = lm(*Attriute we are predicting ~ *first attribute*
                 + *second attribute* +
                 *third attribute* + *fourth attribute* + *fifth attribute*,
                 train)
```

19. Print a summary of your model. Once again, look at the adjusted R-squared value. What does it indicate about the efficacy of your model? Respond as a comment in main.r.

20. Look in the "Coefficients:" section of your outputted summary for your model. R puts asterisks after some of the rows. What do these asterisks represent? Which attributes have asterisks next to them? Respond as a comment in main.r.

21. Make predictions about survival using your model (you may want to reference step 15). You can use this line to only print the actual data and your predicted data side by side:

```
print(test[, c("Survived", "prediction")])
```

What is wrong with your predictions? Why is this happening? Respond as a comment in main.r.

22. Let's remedy this error. Add another column to your test data frame called rounded_pred. As a simple solution, we will round any predictions under 0.5 to 0 (representing a passenger who did not survive), and we will round any predictions above 0.5 to 1 (representing a passenger who survived).

```
# Pseudocode: replace everything within asterisks **
test$rounded_pred <- ifelse(round(*column we want to round*) >= 0.5, 1, 0)
print(test[, c("Survived", "rounded_pred")])
```

Why does our rounded_pred column have NA values? Respond as a comment in main.r.

23. Watch this video about confusion matrices. In a 2x2 confusion matrix, what does top right cell represent? Next, watch this video about metrics derived from a confusion matrix. From a confusion matrix, how do we calculate the accuracy of a model? Respond to both of these questions as a comment in main.r.

24. Build your own confusion matrix using the results of your multiple linear regression model. Here is some pseudocode:

```
#Pseudocode: replace everything inside asterisks **
confusion_matrix <- table(*Actual*, *Predictions*)
print(confusion_matrix)
print(nrow(test))
```

Notice how the number of cells in the confusion matrix is less than the number of cells in the test data frame. Why is this? Respond as a comment in main.r.

25. Run your code from step 24 a few times. What do you notice about your confusion matrix?Why do you think this is happening? Hint: the answer is revealed in the directions of step 12. Respond as a comment in main.r.

26. Calculate the accuracy of your model using the formula in step 23. Record your answer as a comment in main.r.

27. Watch this video on Cross Validation. In your own words, why do you think cross validation is useful? Do you think it will solve our problem in step 25? Respond as a comment in main.r.

28. Now manually use fourfold cross validation to compute an aggregated accuracy score for this model.

```
#Pseudocode: replace aeverything in asterisks **
# Make a new model using manual cross validation

tdataClean <- *remove all NA rows from tdata*

# Randomize our tdata
random_indices <- sample(nrow(tdataClean))
randTdata <- tdataClean[random_indices, ]

# Initialize empty vector to store accuracy scores of each
# block in the cross validation
accScores <- c(0,0,0,0)

for (i in 1:4) {
  # Calculate the indices for the test data
  test_indices <- ((i - 1) *
                     (nrow(randTdata) / 4) + 1):(i * (nrow(randTdata) / 4))

  # Split the data into test and train based on the indices
  test <- randTdata[test_indices, ]
  train <- randTdata[-test_indices, ]

  *Build multiple linear regression model using five features* # See step 18

  *Make predictions* # See steps 21 & 22


  # Calculate accuracy score
  confusion_matrix <- table(*Actual*, *Prediction*)
  truePositive <- confusion_matrix[*?*,*?*]
  falsePositive <- confusion_matrix[*?*,*?*]
  falseNegative <- confusion_matrix[*?*,*?*]
  trueNegative <- confusion_matrix[*?*,*?*]
  acc <- (truePositive + trueNegative) / (truePositive + trueNegative
                                          + falsePositive + falseNegative)
  accScores[i] <- acc

}

print(accScores)
```

What is the average accuracy of the model? Do you think we will be able to build a more accurate model? Respond as a comment in main.r.

STOP: Check in with a TA or instructor before continuing.

29. Watch this video on logistic regression. What is the difference between linear and logistic regression? Respond as a comment in main.r.

30. From our linear regression we determined that Age, Sex, and Pclass were the three most statistically significant attributes out of the five attributes that we can control in our time-travelling avatar. Therefore, we are going to make a multiple logistic regression model using just those three attributes. You are pretty much going to use the same code you used in step 28 to compute the accuracy of this model using fourfold cross validation. The only difference is the one line in which you construct the model, which should look like this:

```
# Pseudocode: replace everything in asterisks **
logRegModel = glm(*What we are predicting* ~ *Attribute1* +
                    *Attribute2* + *Attribute3*,
                  *randomized/cleaned dataset*, family = binomial )
```

What is the average accuracy of this model? Do you think we will be able to build a better model? Respond as a comment in main.r.

31. It is now time to use one of your models to determine what attributes to give your time-travelling self to maximize your survival. Use the following pseudocode to test ten different possible combinations of attributes. Pick the one that maximizes your survival odds. What are they?

```
# Create a data frame with attributes you might give yourself
# Modify these attributes and record how different combinations
# affect your likelihood of survival
person_data <- data.frame(Age = *some number*, Sex = *"male" or "female*",
                          Pclass = *some number 1-3*, Fare=100,
                          Embarked = "C").
# Don't modify values for Fare or Embarked, as these attributes were not used
# in your model

# Use the trained model to predict the probabilities of survival
probabilities <- predict(multiplelrm, newdata = person_data, type = "response")

# Retrieve the probability of survival for the person
survival_probability <- probabilities[1]

# Convert the probability to percent likelihood
percent_likelihood <- survival_probability * 100

# Print the percent likelihood of survival
print(percent_likelihood)
```

STOP: Check in with a TA or instructor

32. Pick a dataset of your own from Kaggle. You will import this dataset into your project and then create both a linear and logistic model on any attributes you would like from this dataset.

STOP: Have a TA or instructor approve your dataset and the features that you want to include in your models.

33. Make your linear and logistic models with this new dataset. The steps are exactly the same as they were for the *Titanic* dataset. You may have to do some more data cleaning though. Use the same fourfold cross-validation method you used earlier in the project.

34. Research a new type of model and create it in R using your dataset. There are so many videos on YouTube walking you through how to make different types of models in R. Here are some potential ideas: Decision Trees, K-Nearest Neighbors, Random Forest Pt. 1/ Random Forest Pt. 2.