MTH 325, Lab #7, Spring 2023 Name _____

Instructions: Follow along with the tutorial portion of the lab. Replicate the code examples in R on your own, along with the demonstration. Then use those examples as a model to answer the questions/perform the tasks that follow. Copy and paste the results of your code to answer questions where directed. Submit your response file and the code used (both for the tutorial and part two). Your code file and your lab response file should each include your name inside. Be sure to follow the write-up directions in the Lab Directions file.

We are going to use movie data (from a decade or so ago), to look at how to perform one-hot encoding. The data is in the file **325lab7data1.xlsx**. We begin by loading the data and looking at the variables. Adjust the path to read the file as needed.

```
library(readxl)
data1<-read_excel("R/daemen/325lab7data1.xlsx")
View(data1)
```

The file contains a column of movie names, which we will ignore. We will convert the studio names and genres into dummy variables and then use them to predict total US gross.

```
genres <- unique(data1$Genre)
genres
```

We will first determine the genres we have to work with. This list is 10 elements long. We are going to code the genres by hand. To do this, I am going to let the category Documentary be the default value that is encoded by all zeros. (I selected this because I suspect it is the lowest grossing movie type.) The remaining categories will each get their own dummy variable. We create those columns using the ifelse() function.

```
data1$Horror<-ifelse(data1$Genre=="Horror",1,0)
```

This creates a new column in the data set called Horror, and is coded 1 if the Genre column reads "Horror", and 0 otherwise. We will repeat this for all the remaining genres (except Documentary).

```
data1$Drama<-ifelse(data1$Genre=="Drama",1,0)
data1$Action<-ifelse(data1$Genre=="Action",1,0)
data1$Adventure<-ifelse(data1$Genre=="Adventure",1,0)
data1$Comedy<-ifelse(data1$Genre=="Comedy",1,0)
data1$Romance<-ifelse(data1$Genre=="Romantic Comedy",1,0)
data1$Thriller<-ifelse(data1$Genre=="Thriller/Suspense",1,0)
data1$BlackComedy<-ifelse(data1$Genre=="Black Comedy",1,0)
data1$Musical<-ifelse(data1$Genre=="Musical",1,0)
```

We should verify that all the encoding worked and that every new column has at least one 1 in it. You can use the sum() function to add the 1's in the column. If you get a 0, check for typos. If you get 1 or larger, then it worked.

```
sum(data1$BlackComedy)
```

There are several packages that have dummy variable encoding functions. We are going to use the caret package. Before we do this, though, we'll need to remove the movie names from the data so that they are not encoded as categories.

```
data1<-data1[-1]
```

Then we load the required package (install it if you need to), and since we've already encoded the genre data, I'm also going to remove that column from the data to be transformed.

```
library(caret)
distributor<-data1[-2]
dummy <- dummyVars(" ~ .", data=distributor)
dist <- data.frame(predict(dummy, newdata=data1))
dist
```

If you view the data now—View(dist)—might be easier. You can see that the Distributor column has been replaced by new dummy columns. But you'll notice that the names of the columns are unwieldy at best. The functions in packages need less typing, but they do also give you less control of the operation. You can also check that it produced one column for each distributor, one column more than we need.

```
distributors <- unique(data1$Distributor)
distributors

sum(dist[1:28])
```

If you sum up the values in the first 28 columns (the dummies for the 28 unique distributor names), you see they add up to 211, which is the number of rows of the data. We will need to remove one of the columns before we use the data for regression since 27 of these columns exactly predict the last one. We'll have to choose which one to eliminate since that will be the default value. I decided to delete column 14 for Paramount Vantage since it does both documentaries (our other default category) and other genres.

We now will have a dataset that has 36 dummy variables and one column of numerical data on US box office that we can make predictions on using a traditional linear regression.

We are going to look at two non-parametric regression models (on a single variable): LOESS and Gaussian processes. LOESS is the default regression method in ggplot so it's worth being able to analyze the model. The Gaussian process model is different in enough respects that it's a nice tool to have. We'll use the mtcars data set to build a model of mpg modeled by weight.

First, we'll build the model.

```
data(mtcars)
loess1<-loess(mpg~wt,data=mtcars)
summary(loess1)
```

Note some of the information that comes from the model summary: equivalent parameters allows you to compare the model to a polynomial model, the default span is 0.75, each segment is modeled by

degree 2 segments, the residual standard error. If we plot the graph using ggplot and geom_smooth() will plot this same model by default.

```
ggplot(data=mtcars,aes(x=wt,y=mpg))+geom_point()+geom_smooth()
```

LOESS can be modified and made more or less smooth by adjusting the size of the span (the amount of the data that is used in the model at each point). The smallest span size will depend on the size of the data. Bigger datasets can tolerate smaller spans because they will include more points. But we can still look at a range of span values and see how they impact the results.

```
loess2<-loess(mpg~wt,data=mtcars,span=0.4)
summary(loess2)
loess3<-loess(mpg~wt,data=mtcars,span=0.5)
summary(loess3)
loess4<-loess(mpg~wt,data=mtcars,span=0.6)
summary(loess4)
loess5<-loess(mpg~wt,data=mtcars,span=0.7)
summary(loess5)
loess6<-loess(mpg~wt,data=mtcars,span=0.8)
summary(loess6)
loess7<-loess(mpg~wt,data=mtcars,span=0.9)
summary(loess7)
```

We can see that smaller spans are equivalent to more parameters (higher degree polynomials), and they need not be whole numbers. The residual standard error also changes, and in this case, seems to be the lowest when span = 0.5.

We can plot the curves with ggplot, and to better see each curve, we can color code them, and turn off the error bars.

```
ggplot(data=mtcars,aes(x=wt,y=mpg))+geom_point()+
  geom_smooth(span=0.4,color="red",se=FALSE)+
  geom_smooth(span=0.5,color="blue", se=FALSE)+
  geom_smooth(span=0.6,color="green", se=FALSE)+
  geom_smooth(span=0.7,color="black", se=FALSE)+
  geom_smooth(span=0.8,color="orange", se=FALSE)+
  geom_smooth(span=0.9,color="purple",se=FALSE)
```

We can also adjust the degree parameter, and apply cross-validation (which we'll look at later) to help us choose the hyperparameters that produce the best fit for the data. But, for now, let's look at Gaussian processes.

There are several packages we can use for gaussian process regression (like a dozen), but for this example we'll use a package called RobustGASP. You'll need to install it before we use it. RobustGASP also requires our data to be in a matrix format.

```
library(RobustGaSP)
input<-as.matrix(data.frame(mtcars$wt))
output<-as.matrix(data.frame(mtcars$mpg))
```

For this exam, the weights range between 1 and 6 (they have already been rescaled), so we don't need to rescale them for this. But, if the values were in smaller units, and therefore much larger, we would need to rescale them. It's standard practice to put them in a range of 0 (min) to 1 (max) because there is an exponential function in the background that works best when the values are smaller. When you run the model with these values, it will still run as is, but you'll get a lot of warnings. If you divide the weights by 6, you'll get fewer warnings.

```
model<- rgasp(design = input, response = output, zero.mean="No",nugget.est=TRUE)
model
```

If convergence fails on the initial rgasp function, try running it a second time. The zero mean assumption is also pretty common in Gaussian process regression. Some packages will require you to shift your data by the mean to enforce this property, but here we can just specify that the mean is not zero.

```
testing_input = as.matrix(seq(0.2,0.9,1/100))
model.predict<-predict(model,testing_input)
names(model.predict)
testing_output=higdon.1.data(testing_input)
plot(testing_input,model.predict$mean,type='l',col='blue',
     xlab='input',ylab='output')
polygon( c(testing_input,rev(testing_input)),
         c(model.predict$lower95,rev(model.predict$upper95)),
         col = "grey80", border = FALSE)
lines(testing_input,model.predict$mean,type='l',col='blue')
lines(input, output,type='p')
```

The testing_input is a set of values we can predict the model at so that we can draw the line on a graph. Here, I've use 0.2 to 0.9 as rescales weight values (after dividing by 6). If you did not divide by 6, replace these values with 1.2 and 5.5 as the equivalent ranges. This plotting sequence uses the base-R plot function. We can use ggplot if we wish, but we'd have to convert the testing input and the predictions into a dataframe, along with the errors at each point in order to manually create both the prediction line and the error bands.

If you want to see what GPs do beyond the range of the data, extend the values of your testing input wider. You'll see that the GP model will regress to the mean by default if we don't specify another trend, which can be more useful than what other models do, which is usually to tend to infinity (such as in polynomial models).

To compare this model to others, we can calculate the mean square error like this.

```
mean((model.predict$mean-testing_output)^2)
```

The last thing to do is to do a brief machine learning example using the iris dataset.

```
data(iris)
dataset <- iris
```

Next, we are going to split the data into test and training datasets. The training data set will be 80% of the available data, and we'll leave the remaining 20% to test our models with.

```
validation_index <- createDataPartition(dataset$Species, p=0.80, list=FALSE)
validation <- dataset[-validation_index,]
dataset <- dataset[validation_index,]
```

Normally, it would be best to explore the data before proceeding, but we are going to skip that for now, since we've done these type of summary and visualization tasks before.

We are going to set up a 10-fold cross validation process in order to validate our models.

```
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

In this example, we are going to compare 5 different machine learning models. Some of which we've talked about at least briefly: Linear Discriminant Analysis (LDA), Classification and Regression Trees (CART), k-Nearest Neighbors (kNN), Support Vector Machines (SVM) with a linear kernel, Random Forest (RF).

Before each run, we are going to set a random seed to ensure that the models run the same each time. (We did not set one before the test-train split, but if we want full replication every time, we should do it there, too.)

Let's build our models. You may be prompted to install packages that are needed if you don't have them. Make note of them and install them, then run the commands again.

```
set.seed(7)
fit.lda <- train(Species~., data=dataset, method="lda",
                 metric=metric, trControl=control)
set.seed(7)
fit.cart <- train(Species~., data=dataset, method="rpart",
                  metric=metric, trControl=control)
set.seed(7)
fit.knn <- train(Species~., data=dataset, method="knn",
                 metric=metric, trControl=control)
set.seed(7)
fit.svm <- train(Species~., data=dataset, method="svmRadial",
                 metric=metric, trControl=control)
set.seed(7)
fit.rf <- train(Species~., data=dataset, method="rf",
                metric=metric, trControl=control)
```

We can compare the accuracy of the models to find the best one.

```
results <- resamples(list(lda=fit.lda, cart=fit.cart,
                          knn=fit.knn, svm=fit.svm, rf=fit.rf))
summary(results)
```

Numbers can be hard to interpret, so we can create a dot plot to examine the results.

```
dotplot(results)
```

We can also look at the results of individual models by printing them. For example:

```
print(fit.lda)
```

We can test the models by using testing (validation) set that we held out at the beginning. The example below uses the LDA model.

```
predictions <- predict(fit.lda, validation)
confusionMatrix(predictions, validation$Species)
```

What we want to see is that the accuracy remains similar to the accuracy of the original model. If it's too high or too low, it may be a sign of a problem with the model, or the testing set may be too small.

Tasks:

1. Using the dataset in **325lab7data2.xlsx**, perform one-hot encoding on the Company Type variable. You should remove both the Company Name, and the column with the names of the CEO. Create a new variable of total compensation (the sum of their base salary plus bonus). Then use the encoded dummy variables to predict CEO salaries. Describe your model and how accurate you think it is.

2. Using the data in **325lab7data3.xlsx**, create models of the included date (predict final exam from midterm scores), comparing linear regression to LOESS or Gaussian process regression. Which model does a better job of predicting and why? Create plots of your models for comparison.

3. Using the data in **325lab7data4.xlsx**, choose two of the machine learning classification models from the examples to perform classification on the data, predicting Opinion from the other variables (not the Person column—remove it). You may need to encode some of the other text variables. Describe your procedure, which models you choose, and which performed better. Follow the validation procedures from the lab examples. Produce appropriate graphs or metrics to support your conclusion.

Resources:

1. https://www.statology.org/one-hot-encoding-in-r/
2. https://www.r-bloggers.com/2021/12/how-to-find-unique-values-in-r/
3. https://www.datamentor.io/r-programming/ifelse-function/
4. https://koalatea.io/r-one-hot-encoding/
5. https://ggplot2.tidyverse.org/reference/geom_smooth.html

6. https://www.statology.org/loess-regression-in-r/
7. https://cran.r-project.org/web/packages/RobustGaSP/index.html
8. https://machinelearningmastery.com/machine-learning-in-r-step-by-step/