Model Development with Tidyverse

Estimated Time Needed: **45 min**

Welcome!

In this section, we will examine model development by trying to predict the arrival delay of a flight using our dataset.

A model or estimator can be thought of as a mathematical equation used to predict a value given one or more other values. Relating one or more independent variables or features to dependent variables.

For example, you input a flight's departure delay as the independent variable or feature, the output of the model or dependent variable is the arrival delay. Usually the more relevant data you have the more accurate your model is.

In data analysis, we often use Model Development to help us predict future observations from the data we have.

A Model will help us understand the exact relationship between different variables and how these variables are used to predict the result.

In this lab you will learn about:

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression

Table of Contents

Load Libraries and Data

First, load the tidyverse library.

[ ]:
```
# Load tidyverse
library(tidyverse)
```

The original Airline dataset is hosted on IBM Data Asset eXchange. This sample dataset can be found here.

Now using the subset dataset link, you can load it and store as a dataframe `sub_airline`:

[ ]:
```
# url where the data is located
url <- "https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/lax_to_jfk.tar.gz"

# download the file
download.file(url, destfile = "lax_to_jfk.tar.gz")

# if you run this on your local machine, then can remove tar = "internal"
untar("lax_to_jfk.tar.gz", tar = "internal")

# read_csv only
sub_airline <- read_csv("lax_to_jfk/lax_to_jfk.csv",
                        col_types = cols('DivDistance' = col_number(),
                                         'DivArrDelay' = col_number()))
```

1. Simple Linear Regression

One example of a Data Model that we will be using is **Simple Linear Regression**.

Simple Linear Regression is a method to help us understand the relationship between two variables:

- X: The predictor/independent variable
- Y: The response/dependent variable (that we want to predict)

The result of Linear Regression is a linear function that predicts the response (dependent) variable as a function of the predictor (independent) variable.

Linear Model Function:

$$\hat{Y} = b_0 + b_1 X$$

- b_0 refers to the intercept of the regression line, in other words: the value of Y when X is 0
- b_1 refers to the slope of the regression line, in other words: the value with which Y changes when X increases by 1 unit
- $\hat{Y}$ (y-hat) is the predicted value from the linear model

Fit the data into a linear regression model

First, let's just look at just Alaska Airline (AA) data, so filter the data first. We also filter out the NAs in CarrierDelay because you will use that variable later.

[ ]:
```
# Define dataset with just AA as the Reporting_Airline
aa_delays <- sub_airline %>%
  filter(CarrierDelay != "NA", Reporting_Airline == "AA")

head(aa_delays)
```

For this example, we want to look at how departure delay (DepDelayMinutes) can help us predict arrival delay (ArrDelayMinutes). Using simple linear regression, we will create a linear function with "DepDelayMinutes" as the predictor variable and the "ArrDelayMinutes" as the response variable. You can use base R's function `lm()` to create a linear model.

[ ]:
```
linear_model <- lm(ArrDelayMinutes ~ DepDelayMinutes, data = aa_delays)
```

Summarize the regression model using `summary()`. The output displays the learned coefficients ("Estimate" in the output) of the model, b_0 and b_1 as well as other information about the fitted model.

[ ]:
```
summary(linear_model)
```

We can output a prediction of three new data points.

[ ]:
```
# Input data we use to predict
new_depdelay <- data.frame(
  DepDelayMinutes = c(12, 19, 24))

# Predict the data points
pred <- predict(linear_model, newdata = new_depdelay, interval = "confidence")
pred
```

When we print the `pred` object, we can see that there are 3 columns: fit, lwr and upr. The "fit" column is the prediction results of the inputs. And "lwr" and "upr" are the lower bound and upper bound of the 95% confidence intervals of prediction results. The confidence interval reflects the uncertainty around the mean predictions.

For example, given that the DepDelayMinutes is 12, then the model predicts the ArrDelayMinutes to be 26.38, and we are 95% confident that the interval (21.98, 30.77) captures the true mean arrival delay for this instance.

What is the value of the intercept (b_0) and the Slope (b_1)?

Using the fitted model, `linear_model`, you can grab the attribute `coefficients` using `$`. These coefficients correspond to b_0 (the intercept) and b_1 (the slope and coefficient of DepDealyMinutes).

[ ]:
```
linear_model$coefficients
```

What is the final estimated linear model we get?

As we saw above, we should get a final linear model with the structure:

$$\hat{Y} = b_0 + b_1 X$$

Remember that we are predicting ArrDelayMinutes using DepDealyMinutes. So, plugging in the actual values we get:

$$ArrDelayMinutes = 17.35 + 0.7523 * DepDelayMinutes$$

Question #1 a):

Create a linear function with "CarrierDelay" as the predictor variable and the "ArrDelayMinutes" as the response variable.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼ Click here for the solution.
```
linear_model2 <- lm(ArrDelayMinutes ~ CarrierDelay,
                    data = aa_delays)
```

Question #1 b):

Find the coefficients (intercept and slope) of the model.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼ Click here for the solution.
```
linear_model2$coefficients
```

Question #1 c):

What is the equation of the predicted line. You can use x and yhat or 'CarrierDelay' or 'ArrDelayMinutes'?

You can type you answer here

▼ Click here for the solution.
```
# using X and Y
Yhat = 35.11 + 0.7032 * X

ArrDelayMinutes = 35.11 + 0.7032 * CarrierDelay
```

2. Multiple Linear Regression

What if we want to predict arrival delay minutes using *more than one* variable?

If we want to use more variables in our model to predict arrival delay minutes, we can use Multiple Linear Regression. Multiple Linear Regression is very similar to Simple Linear Regression, but this method is used to explain the relationship between one continuous response (dependent) variable and two or more predictor (independent) variables. Most of the real-world regression models involve multiple predictors. We will illustrate the structure by using two predictor variables, but these results can generalize to any amount of predictor variables:

- Y: Response Variable
- X_1: Predictor Variable 1
- X_2: Predictor Variable 2

The equation is given by

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2$$

where,

- b_0: intercept
- b_1:coefficient of Variable 1
- b_2: coefficient of Variable 2

From the previous lab we know that other good predictors of ArrDelayMinutes could be:

- DepDelayMinutes
- LateAircraftDelay

Let's develop a model using these variables as the predictor variables by fitting the data.

[ ]:
```
mlr <- lm(ArrDelayMinutes ~ DepDelayMinutes + LateAircraftDelay, data = aa_delays)

summary(mlr)
```

What is the value of the intercept (b_0) and the coefficients (b_1, b_2)?

[ ]:
```
mlr$coefficients
```

What is the final estimated linear model that we get?

As we saw above, we should get a final linear function with the structure:

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2$$

What is the linear function we get in this example?

$$ArrDelayMinutes = 17.32 + 0.7556_{D}epDelayMinutes - 0.0103_{L}ateAircraftDelay$$

Question #2 a):

Create and train a Multiple Linear Regression model "mlr2" where the response variable is ArrDelayMinutes, and the predictor variable is 'DepDelayMinutes', 'LateAircraftDelay' and 'CarrierDelay'.

[ ]:
```
# Write your code below and press Shift+Enter to execute

```

▼Click here for the solution.

```
mlr2 <- lm(
        ArrDelayMinutes ~ DepDelayMinutes +
        LateAircraftDelay + CarrierDelay,
        data = aa_delays)

summary(mlr2)
```

Question #2 b):

Find the coefficients of the model?

[ ]:
```
# Write your code below and press Shift+Enter to execute

```

▼Click here for the solution.

```
mlr2$coefficient
```

Question #2 c):

Using the fitted model, mlr2, what are the predicted values for the following new data points?

[ ]:
```
# New data points
DepDelayMinutes <- c(10, 20, 30)
LateAircraftDelay <- c(20, 60, 30)
new_multidelay <- data.frame(DepDelayMinutes, LateAircraftDelay)
```

[ ]:
```
# Write your code below and press Shift+Enter to execute

```

▼Click here for the solution.

```
pred <- predict(mlr,
        newdata = new_multidelay,
        interval = "confidence")
```

3. Assessing Models Visually

Now that we've developed some models, how do we evaluate our models and how do we choose the best one? One way to do this is by using visualization.

Regression Plot

When it comes to simple linear regression, an excellent way to visualize the fit of our model is by using regression plots.

Regression plots are a good estimate of:

- The relationship between two variables,
- The strength of the correlation, and
- The direction of the relationship (positive or negative).

There are several ways to plot a regression plot; a simple way is to use "ggplot" from the tidyverse library.

This plot will show a combination of a scattered data points (a scatter plot), as well as the fitted linear regression line going through the data. This will give us a reasonable estimate of the relationship between the two variables, the strength of the correlation, as well as the direction (positive or negative correlation).

Let's visualize the DepDelayMinutes and ArrDelayMinutes of Alaska Airlines subset dataframe which we created earlier. Recall from the previous lab that to visualize a fitted linear model using `ggplot`, you can use `geom_smooth()`. Additionally, you can use `stat_smooth()` to create the same model. Like before, the default model if unspecified is `formula = y ~ x`. In this case, you will predict arrival delay minutes using depature delay minutes. So the predictor variable is "DepDelayMinutes" and the response variable is "ArrDelayMinutes".

[ ]:
```
ggplot(aa_delays, aes(x = DepDelayMinutes, y = ArrDelayMinutes)) +
  geom_point() +
  stat_smooth(method = "lm", col = "red")
```

We can see from this plot that Arrival Delay Minutes (ArrDelayMinutes) is positively correlated to Departure Delay Minutes (DepDelayMinutes), since the regression slope is positive.

One thing to keep in mind when looking at a regression plot is to pay attention to how scattered the data points are around the regression line. This will give you a good indication of the variance of the data, and whether a linear model would be the best fit or not. If the data is too far off from the line, this linear model might not be the best model for this data.

Question #3 a):

Create a regression plot of "CarrierDelay" and "ArrDelayMinutes" using "aa_delays" dataset

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
ggplot(
    aa_delays,
    aes(x = CarrierDelay, y = ArrDelayMinutes)) +
  geom_point() +
  stat_smooth(method = "lm", col = "red")
```

Question #3 b):

Given the regression plots above is "DepDelayMinutes" or "CarrierDelay" more strongly correlated with "ArrDelayMinutes". Use the method "cor()" to verify your answer.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
# The variable "DepDelayMinutes" has a stronger correlation with "ArrDelayMinutes", it is approximately 0.871  compared to "CarrierDelay" which is approximately 0.624. You can verify it using the following commands:

cor(aa_delays@@0@@@ArrDelayMinutes)
cor(aa_delays@@1@@@ArrDelayMinutes)
```

Residual Plot

A good way to visualize the variance of the data is to use a residual plot. Before we start creating residual plots let's first answer the following questions:

- What is a **residual**?
  - The difference between the observed value (Y) and the predicted value ($\hat{Y}$) is called the residual (or error). When we look at a regression plot, the residual is the distance from the data point to the fitted regression line.
- What is a **residual plot**?
  - A residual plot is a graph that shows the residuals on the vertical y-axis and the independent variable on the horizontal x-axis.
- What do we pay attention to when looking at a residual plot?
  - **Homoscedasticity**: If the residual plot is homoscedastic, then the points in the plot are **randomly spread out around the x-axis**, which means that a **linear model is appropriate** for the data. Why is that? Randomly spread out residuals means that the variance is constant, and thus the linear model is a good fit for this data.

Now, let's look again at the regression plot of ArrDelayMinutes as the response and DepDelayMinutes as the predictor. This time, let's visualize the residuals on this plot.

- The red line is the regression line
- The black dots represent the observed values of ArrDelayMinutes
- The white dots are the predicted values from the linear regression model
- The light gray lines are the residuals, or errors. It shows how far away the observed values are from the predicted values. So a longer line means a larger error.

[ ]:
```
aa_delays <- sub_airline %>%
  filter(CarrierDelay != "NA", Reporting_Airline == "AA")
score_model <- lm(ArrDelayMinutes ~ DepDelayMinutes, data = aa_delays)
aa_delays$predicted <- predict(score_model)

ggplot(aa_delays, aes(x = DepDelayMinutes, y = ArrDelayMinutes)) +
  geom_smooth(method = "lm", se = FALSE, color = "red") +  # Plot regression slope
  geom_segment(aes(xend = DepDelayMinutes, yend = predicted), alpha = .2) +  # alpha to fade lines
  geom_point() +
  geom_point(aes(y = predicted), shape = 1) +
  theme_bw()  # Add theme for cleaner look
```

Next, you can create a residual plot, which graphs the residuals (light gray lines in the previous graph) against the observed DepDelayMinutes. The code to do this is similar to a normal scatterplot, but you pass in the linear model `lm(ArrDelayMinutes ~ DepDelayMinutes)` and when setting the y axis, you can use `.resid` which will use the residuals from the model you inputted.

We can see from this residual plot that the residuals are not randomly spread around the x-axis, which leads us to believe that maybe a non-linear model is more appropriate for this data.

[ ]:
```
ggplot(lm(ArrDelayMinutes ~ DepDelayMinutes, data = aa_delays)) +
  geom_point(aes(x=DepDelayMinutes, y=.resid))
```

Other Diagnostic Plots

In addition to residual plots, there are other useful plots. A simple way to view diagnostic plots is to first create the linear model using `lm()`, then call base R's `plot()` function on the model.

The below code will output four graphs:

1. **Reisudal plot**: Identical to the graph we made with ggplot, here it again shows that the residuals are not randomly spread around the x-axis.
2. **Q-Q plot**: The dotted diagonal line represents what normally distributed error (residual) values would follow. In this case, the residuals do not look normally distributed since there are many observations that fall above the line on the right side.
3. **Scale-location plot**: This plot helps check the homoscedasticity assumption. Here, it shows a red line that is not straight and validates the homoscedasticity assumption is not satisfied.
4. **Residuals vs leverage plot**: This helps determine **influential points**. Any points outside the dotted lines (Cook's distance) would make it influential. Here, none of the points cross the lines, however several points come close and could be removed or analyzed further.

[ ]:
```
linear_model <- lm(ArrDelayMinutes ~ DepDelayMinutes, data = aa_delays)
plot(linear_model)
```

4. Polynomial Regression

Polynomial regression is a particular case of the general linear regression model or multiple linear regression models. That is, although the data is nonlinear in polynomial regression (the predicator variables have higher order terms), the model in all cases is **linear**. The model is always **linear** because it predicts the coefficients ($b_0, b_1, \ldots$) which are always of order 1.

There are different orders of polynomial regression:

Quadratic - 2nd order
$$Y = b_0 + b_1 X + b_2 X^2$$

Cubic - 3rd order
$$Y = b_0 + b_1 X + b_2 X^2 + b_3 X^3$$

Higher (n^{th}) order:
$$Y = b_0 + b_1 X + b_2 X^2 + b_3 X^3 \ldots + b_n X^n$$

Let's look at the below example. Here, we create random predictor variable `q` and random response variable `y` that follows a 3rd order polynomial but then we add some random noise to it to get `noise.y`. We set the seed so that this result can be reproduced.

[ ]:
```
set.seed(20)
x <- seq(from=0, to=20, by=0.1)

# value to predict (y):
y <- 500 + 0.4 * (x-10)^3

# some noise is generated and added to the real signal (y):
noise <- rnorm(length(x), mean=10, sd=80)
noisy.y <- y + noise
```

In the graph below, we fit a first order linear model to this example dataset and can see that the model does not fit the data very well.

[ ]:
```
# fit linear model
ggplot(data=NULL,aes(x, noisy.y)) +
    geom_point() +
    geom_smooth(method = "lm")
```

Instead, we can use a polynomial model. It is similar to the first order linear model except that you include `poly()` within `geom_smooth()` to indicate what order polynomial to use. For example, using `poly(x, 5)` equates to having $b_0 + b_1 X^2 + b_2 X^2 + b_3 X^3 + b_4 X^4 + b_5 X^5$.

[ ]:
```
ggplot(data=NULL,aes(x, noisy.y)) +
    geom_point() +
    geom_smooth(method = "lm", formula = y ~ poly(x, 5))
```

We can already see from plotting that this polynomial model performs better than the linear model. This is because the generated polynomial function "hits" more of the data points.

Polynomial 2nd Order

Now let's look at another example, this time using a 2nd order polynomial. Again, we use a toy dataset where `time` is the predictor and `temp` is the response.

[ ]:
```
time <- 6:19
temp <- c(4,6,7,9,10,11,11.5,12,12,11.5,11,10,9,8)

ggplot(data = NULL, aes(time, temp)) +
    geom_point()
```

We can create a model like how we saw before using `lm()` and to include higher order, you can used `poly()`.

For this dataset, we try a 2nd order polynomial model to see how it fits. The equation the model follows is:

temp = b_0 + b_1 _ time + b_2 _ time^2

[ ]:
```
polyfit2 <- lm(temp ~ poly(time, 2, raw = TRUE))

summary(polyfit2)
```

From the summary output of the model, you can find the coefficients, so to predict temp, you could use:
$$temp = -13.7 + 3.76_t ime - 0.138_t ime^2$$

Like for the first order linear models, you can use `ggplot` to graph the model.

[ ]:
```
ggplot(data = NULL, aes(time, temp)) +
    geom_point() +
    geom_smooth(method = "lm", formula = y ~ poly(x, 2))
```

Question #4 a):

Create a 4th order polynomial model with the variables time and temp from above and display the summary of the model.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼ Click here for the solution.

```
# calculate polynomial
polyfit4 <- lm(temp ~ poly(time, 4, raw = TRUE))
# print results
summary(polyfit4)
```

Question #4 b):

Using the predicted coefficients from the summary output for the 4th order model, write down the model equation.

You can type your answer here.

▼ Click here for the solution.

```
temp = 0.9580 -1.683 * time
    + 0.5770 * time^2
    - 0.03971 * time^3
    + 0.0007906 * time^4
```

5. Assessing the Model

When evaluating our models, not only do we want to visualize the results, but we also want a quantitative measure to determine how accurate the model is.

Two very important measures that are often used in Statistics to determine the accuracy of a model are:

1. R^2 / R-squared
2. Mean Squared Error (MSE)

R-squared

R squared, also known as the coefficient of determination, is a measure to indicate how close the data is to the fitted regression line. The value of the R-squared is the percentage of variation of the response variable (y) that is explained by a linear model.

Mean Squared Error (MSE)

$$MSE = average((\hat{y} - y)^2)$$

$$RMSE = \sqrt{MSE}$$

The Mean Squared Error measures the average of the squares of errors, that is, the difference between actual value (y) and the estimated value (ŷ). Another metric that is related to MSE is **root mean squared error (RMSE)** and is simply the square root of MSE.

Model 1: Simple Linear Regression

Let's use the simple linear regression model we created in section 2.

[ ]:
```
linear_model <- lm(ArrDelayMinutes ~ DepDelayMinutes, aa_delays)
```

Using this model, you can calculate MSE and RMSE. From below, MSE is 394 and RMSE is 19.85.

[ ]:
```
mse <- mean(linear_model$residuals^2)
mse
```

[ ]:
```
rmse <- sqrt(mse)
rmse
```

R^2 can be obtained from the summary of the model. From the output below, we can say that approximately 75.9% of the variation of price is explained by this simple linear model.

[ ]:
```
summary(linear_model)$r.squared
```

Model 2: Multiple Linear Regression

Next, let's use the multiple linear regression model we created in section 3.

[ ]:
```
mlr <- lm(ArrDelayMinutes ~ DepDelayMinutes + LateAircraftDelay, data = aa_delays)
```

Let's calculate MSE and RMSE. From below, MSE is 394 and RMSE is 19.849.

[ ]:
```
mse_mlr <- mean(mlr$residuals^2)
mse_mlr
```

[ ]:
```
rmse_mlr <- sqrt(mse_mlr)
rmse_mlr
```

From the r-squared value belwo, we can say that approximately 75.9 % of the variation of Arrival Delay Minutes is explained by this multiple linear regression "mlr".

[ ]:
```
summary(mlr)$r.squared
```

Model 3: Polynomial Regression

Finally, we can use a polynomial regression model using the skills from section 5.

[ ]:
```
poly_reg <- lm(ArrDelayMinutes ~ poly(DepDelayMinutes, 3), data = aa_delays)
```

Similar to model 1 and 2, you can find MSE, RMSE, and R^2. Here the MSE is 328.97, RMSE is 19.85, and R^2 is 0.798.

[ ]:
```
mse_poly <- mean(poly_reg$residuals^2)
mse_poly
```

[ ]:
```
rmse_poly <- sqrt(mse)
rmse_poly
```

[ ]:
```
summary(poly_reg)$r.squared
```

6. Prediction and Decision Making

Prediction

Previously, we trained the model using the method `lm()` and we used the method `predict()` to produce a prediction.

[ ]:
```
# For example we want to predict the score model we created in a previous section
head(predict(score_model))
```

Decision Making: Determining a Good Model Fit

Now that we have visualized the different models, and generated the R-squared and MSE values for the fits, how do we determine a good model fit?

- What is a good **R-squared** value?
  - When comparing models, the model with the *higher R-squared* value is a better fit for the data.
- What is a good **MSE**?
  - When comparing models, the model with the *smallest MSE* value is a better fit for the data.

Let's take a look at the values for the different models.

Simple Linear Regression: Using DepDelayMinutes as a Predictor Variable of ArrDelayMinutes.

- R-squared: 0.7588
- MSE: 394.063

Multiple Linear Regression: Using DepDelayMinutes and LateAircraftDelay as Predictor Variables of ArrDelayMinutes.

- R-squared: 0.75883
- MSE: 394.0113

Polynomial Fit: Using 3rd Oder Polynomial of DepDelayMinutes as a Predictor Variable of ArrDelayMinutes

- R-squared: 0.7986
- MSE: 328.9701

Simple Linear Regression model (SLR) vs Multiple Linear Regression model (MLR)

Usually, the more variables you have, the better your model is at predicting, but this is not always true. Sometimes you may not have enough data, you may run into numerical problems, or many of the variables may not be useful and or even act as noise. As a result, you should always check the MSE and R^2.

So to be able to compare the results of the MLR vs SLR models, we look at a combination of both the R-squared and MSE to make the best conclusion about the fit of the model.

- MSE: The MSE of SLR model is 394.063 while MLR has an MSE of 394.0113. The MSE of MLR model is ever slightly smaller.
- R-squared: In this case, we can see that the R-squared for the SLR is a little lower than the R-squared for the MLR model.

This R-squared in combination with the MSE show that MLR seems like a slightly better model fit in this case, compared to SLR. However, you could try adding more predictor variables in the MLR model to see if that made a bigger improvement since in our example only two were used.

Simple Linear Model (SLR) vs Polynomial Fit

- MSE: We can see that Polynomial model brought down the MSE, since this MSE is smaller than the one from the SLR.
- R-squared: The R-squared for the Polyfit is larger than the R-squared for the SLR, so the Polynomial Fit also brought up the R-squared quite a bit.

Since the Polynomial Fit resulted in a lower MSE and a higher R-squared, we can conclude that this was a better fit model than the simple linear regression for predicting ArrDelayMinutes.

Multiple Linear Regression (MLR) vs Polynomial Fit

- MSE: The MSE for the polynomial model is smaller than the MSE for the MLR model.
- R-squared: The R-squared for the polynomial model is also larger than the MLR model's.

Conclusion:

Comparing these three models, the MLR model performs slightly better than the SLR model. Perhaps if we tried adding some more predictor variables, the MLR model could do even better. Of the three models, we conclude that the *polynomial of order 3 model* seems to be the best fit it as it has the highest R^2 and the lowest MSE.

As a bonus, you can try using more predictor variables and different order polynomials to perhaps find even better results.

Thank you for completing this notebook

About the Authors:

This notebook was written by Yiwen Li and Gabriela de Queiroz.

Yiwen Li is a developer advocate and data scientist at IBM. She has been creating online content such as code patterns, tutorials, and blogs demonstrating the potential of products and services offered by IBM (like Watson Studio, Machine learning, Model Asset eXchange, Data Asset eXchange, etc.). She holds dual degree, including BS in Probability and Statistics and BA in Economics from the University of California, San Diego.

Gabriela de Queiroz is a Sr. Engineering & Data Science Manager at IBM where she manages and leads a team of developers working on Data & AI Open Source projects. She works to democratize AI by building tools and launching new open source projects. She is the founder of AI Inclusive, a global organization that is helping increase the representation and participation of gender minorities in Artificial Intelligence. She is also the founder of R-Ladies, a worldwide organization for promoting diversity in the R community with more than 190 chapters in 50+ countries. She has worked in several startups and where she built teams, developed statistical models, and employed a variety of techniques to derive insights and drive data-centric decisions.


Simple
0
1
R | Idle
Initialized (additional servers needed)
Mem: 229.85 / 6144.00 MB
Mode: Command
Ln 1, Col 1
English (American)
DA0151EN-Review-Model-Development.ipynb