Overview

This R Shiny application, named "Data Science Curriculum Explorer", enables users to explore and analyze curriculum. Users can select courses based on their year and subject code, visualize course dependencies, analyze the structural complexity of a given curriculum, get predicted course grades, and receive personalized course recommendations. The application integrates with an SQLite database for persistent storage and utilizes various R packages for data manipulation, visualization, and interaction.

Raw Shiny code can be difficult to understand, so this short document's purpose is to explain how each section works and where to go if you want to change something. I assume some basic familiarity with Shiny in this guide. If you're a student, reading up to the reactivity section of an intro Shiny textbook should be sufficient.

If you have any questions email me at danielkrasnovdk@hotmail.com.

Background and Quick Shiny Refresher

This app depends on a few libraries:

- **CurricularAnalytics:** functions for generating Curricular Analytics graph metrics.
- **visNetwork:** Creation of interactive network graph visualizations.
- **shiny:** Building interactive web apps.
- **reticulate:** Interfacing with Python (for topic model).
- **randomForest:** Fitting random forest models (for student grade prediction).
- **stringr:** String manipulation.
- **dplyr:** Data manipulation.
- **readr:** Reading CSV files.
- **DBI:** Database Interface for communication with databases. Courses selected are stored in a database and updates as you add and drop them.
- **RSQLite:** SQLite interface for R.

Shiny apps are comprised of two functions: `ui` and `server`. The `ui` function defines all ui components and the structure of the app. The `server` function holds all the logic of the app and decides what the ui components look like. In the server `function`, ui components are referenced through `input$<component_name>`. For example if in the `ui` function I have `actionButton("resetButton", "Reset")` then that means I could go `input$resetButton` in the `server` function to define behavior for this ui component. An exception to this rule is any `uiOutput()` function. This refers to a piece of ui that changes based on server logic. For example if in the `ui` function I have `uiOutput("coursestaken")` then this ui would be generated according to the server logic inside of `output$coursestaken <- renderUI({ … })`.

If anything in the app is still confusing after reading this guide and looking at the code. I recommend asking ChatGPT for help understanding the app and making changes. I find it is fairly good at writing Shiny code. Otherwise feel free to email me as well.

UI

The UI functions and named components in the app are:

- **selectInput:** dropdowns for selecting year and subject code.
  - Named components are `dropdownYear`, `dropdownCourseCode`, and `sugYear`.
- **uiOutput:** dynamic UI elements:
  - Named components are `pred_grad_output`, `coursestaken` and `courseRecSim`.
- **actionButton:** buttons for resetting selections and submitting inputs.
  - Named components are called `resetButton`, `submit_button`, and `submit_button_pred_course_grad`.
- **visNetworkOutput:** output for the interactive network graph:
  - Named component is `network`.
- **textAreaInput:** text inputs for grade prediction and course similarity analysis.
  - Named components are called `response_input`, `predictor_input`, `text_input` and `predictor_grade_input`

Admittedly, these names are not very descriptive, however it should be obvious what these components do by looking at where they are in the UI or by reading the descriptions below.

Server

The `server` function of the app is what controls the logic. In general, the app works according to these 4 steps:

1. Create a database to hold selected courses (if one doesn't already exist).
2. Draw a graph of whatever selected courses there are.
3. Monitor any changes to the database to update the checkboxes and display the curriculum graph.
4. Show predicted course grades or recommended courses if the appropriate options are selected in the right panel.

Raw Shiny code can be difficult to read because functions are not run linearly. That is, due to reactivity, any functions can be called at any time making it difficult to track the flow of the program. I have organized the code into 5 sections using comments. Each section is explained below.

Database Section

This section of the code is responsible for creating, updating, and destroying the database. Courses that have been selected in the app are recorded and stored. This reason we did not simply use a variable is that Shiny will destroy any variable not being used constantly. This way we can get persistent storage while using the app and the user can pick up where they left off after closing the app. I will go through each block of code in order.

```
databaseContents <- reactivePoll(
  50,
  session,
  checkFunc = function() {
    file.info("my_courses_db2.sqlite")$mtime
  },
  valueFunc = function() {
    dbGetQuery(db, "SELECT * FROM selected_courses")
```

```
  }
)
```

- `reactivePoll` checks every 50 ms if the database has changed and if it has it updated the variable `databaseContents`. This variable is what is checked in the graphing section to see what courses to plot.

```
db <-
  dbConnect(RSQLite::SQLite(), "my_courses_db2.sqlite")

# If table does not exist create it
query <-
  "CREATE TABLE IF NOT EXISTS selected_courses (
  id INTEGER PRIMARY KEY,
  course_name TEXT,
  year INTEGER,
  course_code TEXT)"
dbExecute(db, query)
```

- This code simply initiates a connection to the database file and if no table exists creates one. `dbConnect()` will create the file `my_courses_db2.sqlite` if it does not already exist.

```
# Helper function to update database based on checkbox changes
updateDatabaseBasedOnCheckbox <-
  function(inputId, year, courseCode) {
    # Get current selections from the input
    selectedCourses <- input[[inputId]]

    # Fetch current selections from the database for this year and course code
    currentSelections <-
      dbGetQuery(
        db,
        sprintf(
          "SELECT course_name FROM selected_courses WHERE year = %d AND course_code = '%s'",
          year,
          courseCode
        )
      )

    # Determine courses to add or remove
    coursesToAdd <-
      setdiff(selectedCourses, currentSelections$course_name)
    coursesToRemove <-
      setdiff(currentSelections$course_name, selectedCourses)

    # Insert new selections
    sapply(coursesToAdd, function(course) {
      dbExecute(
        db,
```

```
        "INSERT INTO selected_courses (course_name, year, course_code) VALUES (?, ?, ?)",
        params = list(course, year, courseCode)
      )
    })

    # Remove deselected courses
    sapply(coursesToRemove, function(course) {
      dbExecute(
        db,
        "DELETE FROM selected_courses WHERE course_name = ? AND year = ? AND course_code = ?",
        params = list(course, year, courseCode)
      )
    })

    currentCourses <-
      dbGetQuery(db, "SELECT course_name FROM selected_courses WHERE year = 1")
    cat(
      "Courses selected for Year 1:",
      paste(currentCourses$course_name, collapse = ", "),
      "\n"
    )
  }

# Observe changes to checkboxes and update the database accordingly
observe({
  updateDatabaseBasedOnCheckbox("checkboxes1", 1, input$dropdownCourseCode)
})

observe({
  updateDatabaseBasedOnCheckbox("checkboxes2", 2, input$dropdownCourseCode)
})

observe({
  updateDatabaseBasedOnCheckbox("checkboxes3", 3, input$dropdownCourseCode)
})

observe({
  updateDatabaseBasedOnCheckbox("checkboxes4", 4, input$dropdownCourseCode)
})
```

- `updateDatabaseBasedOnCheckbox` is what updates the database of selected courses based on what checkboxes you click on the left panel when using the app. It simply takes the set difference of what has been clicked and what is already in the databases and does the appropriate SQL queries to update the database.
- `observe({...})` is what checks if the corresponding checkbox has been changed and if so calls `updateDatabaseBasedOnCheckbox`. There are 4 dropdown menus in the `uiOutput` called `coursestaken` which are explained later. That is where the ids `checkboxes` 1 through 4 comes from.

```
# Logic for reset button to remove all courses from database
observeEvent(input$resetButton, {
```

```
  # Reset the UI elements for all checkboxes to be unselected
  updateCheckboxGroupInput(session, "checkboxes1", selected = character(0))
  updateCheckboxGroupInput(session, "checkboxes2", selected = character(0))
  updateCheckboxGroupInput(session, "checkboxes3", selected = character(0))
  updateCheckboxGroupInput(session, "checkboxes4", selected = character(0))

  # Execute a query to delete all records from the 'selected_courses' table in the database
  dbExecute(db, "DELETE FROM selected_courses")
})
```

- The reset button on the left pane of the app is controlled by this function. `observeEvent` checks if the ui component `resetButton` has been clicked and if it has, all checkboxes are unselected and the database is emptied out.

Graphing Section

This section of code is for the main graph in the center pane of the app.

```
# Render curriculum graph
 output$network <- renderVisNetwork({
   predictor_input <-
     databaseContents() # Courses are taken from database

   cat("Selected courses in graph:\n")
   cat(paste(predictor_input$course_name,sep = ","))
   cat("\n")

   # If no courses selected, display an empty graph
   if (nrow(predictor_input) == 0) {
     empty_nodes <-
       data.frame(id = numeric(0), label = character(0))
     empty_edges <-
       data.frame(from = numeric(0), to = numeric(0))
     visNetwork(empty_nodes, empty_edges)
   } else {
     courseNames <- predictor_input$course_name
     plot_graph(courseNames)
   }
 })
```

- This function controls the middle pane output and is what shows the curriculum graph. As explained above, `databaseContents` is polled every 50 ms and if a change has occurred this function will be called as well. If no courses have been selected, `visNetwork()` is called with empty arguments to create a placeholder graph. Otherwise, `plot_graph()` is called to display the graph. The logic for `plot_graph()` is in `functions.R` and its explanation is omitted. It is essentially a copy of the `plot_curriculum_graph()` function from the `CurricularAnalytics` package. Feel free to reference the documentation for this package for more information.

Course Navigation Section

This section of code is for the drop down menus and checkboxes for selecting courses.

```
output$coursestaken <- renderUI({
    # Get the year number for the selected Year
    yearNum <-
      as.numeric(gsub("Year ", "", input$dropdownYear))

    # Fetch selected courses from the database for the current year
    selectedCourses <-
      dbGetQuery(db,
                 paste0(
                   "SELECT course_name FROM selected_courses WHERE year = ",
                   yearNum
                 ))

    # Extract the course names to a vector
    selectedCourseNames <-
      selectedCourses$course_name

    # Now use selectedCourseNames for the 'selected' parameter to maintain previosuly selected
courses
    if (input$dropdownYear == "Year 1") {
      checkboxGroupInput(
        "checkboxes1",
        "Choose Options",
        choices = getCourses(
          1,
          "../data/Example-Curriculum.csv",
          input$dropdownCourseCode
        ),
        selected = selectedCourseNames
      )
    } else if (input$dropdownYear == "Year 2") {
      checkboxGroupInput(
        "checkboxes2",
        "Choose Options",
        choices = getCourses(
          2,
          "../data/Example-Curriculum.csv",
          input$dropdownCourseCode
        ),
        selected = selectedCourseNames
      )
    } else if (input$dropdownYear == "Year 3") {
      checkboxGroupInput(
        "checkboxes3",
        "Choose Options",
```

```
        choices = getCourses(
          3,
          "../data/Example-Curriculum.csv",
          input$dropdownCourseCode
        ),
        selected = selectedCourseNames
      )
    } else if (input$dropdownYear == "Year 4") {
      checkboxGroupInput(
        "checkboxes4",
        "Choose Options",
        choices = getCourses(
          4,
          "../data/Example-Curriculum.csv",
          input$dropdownCourseCode
        ),
        selected = selectedCourseNames
      )
    }
  })
```

- `coursestaken` is a reactive ui component that changes depending on what is selected. `input$dropdownYear` is the year the user selects in the menu. Then we query the database for courses in that year using `getCourses()`. This function is defined in `functions.R` and is just a regex to show all course codes available in that year. Then another dropdown menu is created using `checkboxGroupInput()` where all courses for the year and subject selected are shown. The argument `selected = selectedCourseNames` is what keeps previously selected course boxes checked as you navigate to other menus.

Course Prediction Section

This section is still a work in progress and will need to be completely overhauled once a predictive model for course grades is created. For now, the code trains a random forest with response and predictors specified in the app by the user.

```
course_pred_data <- eventReactive(input$submit_button_pred_course_grad,{ df <-
    read.csv("..\\data\\student-data.csv")
  df_clean <- df[,c(1,12,13,15,16)]
  df_clean <- na.omit(df_clean)
  # Do some cleaning on chr columns
  df_clean$STUD_NO_ANONYMOUS <- trimws(df_clean$STUD_NO_ANONYMOUS)
  df_clean$CRS_DPT_CD <- trimws(df_clean$CRS_DPT_CD)
  df_clean$HDR_CRS_LTTR_GRD <- trimws(df_clean$HDR_CRS_LTTR_GRD)

  # Factor grades column
  grades <-
    c("A+", "A", "A-", "B+", "B", "B-", "C+", "C", "C-", "D", "F")
  df_clean$HDR_CRS_LTTR_GRD <-
    factor(df_clean$HDR_CRS_LTTR_GRD, levels = grades)
```

```
  # Create course code column
  df_clean$COURSE_CODE <- paste(df_clean$CRS_DPT_CD, df_clean$CRS_NO, sep = ".")
  df_clean <- df_clean[,-(2:3)]
  df <- subset(df, HDR_CRS_PCT_GRD < 999)
  df_clean <- subset(df_clean, HDR_CRS_PCT_GRD < 999)
  df_clean})
```

- `course_pred_data` is a reactive value to clean and prepare the data. This is very inefficient and was just a place holder while we changed the predictive model and dataset often. This should be changed to reading a csv of cleaned data when the predictive model is finalized. The data is a `eventReactive` on `input$submit_button_pred_course_grad` which means when you press the submit button with this id the cleaned dataset is generated.

```
# Initialize a reactive value to control UI display
  values <- reactiveValues(ready = FALSE)

  observeEvent(input$submit_button_pred_course_grad, {
    cat("Loading data for grade prediction.\n")
    data <- course_pred_data()  # Loading your data
    cat("Data loaded.\n")

    # Handling user inputs
    response <- input$response_input
    predictor_input <- input$predictor_input
    predictor_input <- trimws(unlist(strsplit(predictor_input, ",")))
    predictor_input <- c(predictor_input, response)
    cat("Here is the predictor input:\n")
    cat(predictor_input)
    cat("\n")

    cat("Here is the response input:\n")
    cat(response)
    cat("\n")

    # Preparing data for prediction
    empty <- setNames(as.data.frame(matrix(nrow = 1, ncol = length(predictor_input), data =
NA)), predictor_input)

    # This is just for the conference, change this later
    if(sum(c("DATA.311","STAT.230","MATH.101","COSC.111","MATH.100") %in% predictor_input) == 5)
{
      load("../data/grade.RData")
      preds <- format_data_no_fail_handling.out
      cat("Empty loaded sucessfully\n")
    }else{
      preds <- format_data_no_fail_handling(empty, predictor_input, data)
    }
```

```
  # Running the random forest model
  cat("Starting rf\n")
  rf.out <- randomForest(as.formula(paste0(response,"~.")), data=preds)

  # Extracting importance and determining the most important course
  impor <- importance(rf.out)
  most_important_course <- rownames(impor)[which.max(impor)]

  # Preparing prediction input data
  pred_courses_scores <- input$predictor_grade_input
  pred_courses_scores <- as.numeric(unlist(strsplit(pred_courses_scores, ",")))
  out_mat <- matrix(pred_courses_scores)
  rownames(out_mat) <- predictor_input[predictor_input != response]
  out_mat <- t(out_mat)
  out_df <- as.data.frame(out_mat)

  # Predicting course grades
  predict.out <- predict(rf.out, newdata=out_df)

  # Update reactive values
  output$predicted_grades <- renderTable({
    data.frame(Course = names(predict.out), Predicted_Grade = as.numeric(predict.out))
  })

  output$most_important_course_output <- renderText({
    most_important_course
  })

  # Set reactive value to TRUE to indicate that processing is complete and UI should update
  values$ready <- TRUE
})
```

- `values` is an indicator to display the ui in the following code block.
- Next is an `observeEvent` on `input$submit_button_pred_course_grad`. This means when this button is pressed data is loaded via `course_pred_data()` and a random forest is trained with `randomForest()`. Random Forests have a variable importance measure which is retrieved using `importance()`. Then a prediction is made with the provided courses grades in `input$predictor_grade_input` using `predict(rf.out, newdata=out_df)`. Finally `values$ready <- TRUE` indicates the prediction is done and the next code block is run.

```
# Render UI components conditionally based on the reactive value
output$pred_grad_output <- renderUI({
  if (values$ready) {
    tagList(
      h4("Predicted Course Grades:"),
      tableOutput("predicted_grades"),
      h4("Most Important Course for Prediction:"),
      textOutput("most_important_course_output")
    )
```

```
  }
})
```

- This block renders the `uiOutput` with id `pred_grad_output`. It first checks if `values$ready == TRUE` from the previous block and prints a tale of results.

Course Recommendation Section

This section of code is for the course recommendation engine developed in Python. It first checks if a Python environment is available with the necessary packages and if so, trains and retrieves the document embeddings from a topic model. A much better, more efficient approach, would be to have the document embeddings already in a csv and simply take the cosine similarity to see which courses are most similar to the course of interest. At the time, we were experimenting with different models which is why it is coded like this.

```
reactive_data <-
  eventReactive(input$submit_button, {
    ti <- input$text_input
    sg <- input$sugYear

    if (!is.null(ti) &
        !is.null(sg) & nchar(ti) > 0) {
      cat("Attempting to create Python environment.\n")
      use_or_create_env()

      df <-
        read_csv(
          "..\\data\\UBCO_Course_Calendar.csv",
          locale = locale(encoding = "ISO-8859-1")
        ) |>
        filter(!is.na(`Course Description`))


      cat("Staring doc sim.\n")
      lsaDocSim.out <- lsaDocSim(ti, sg, df)
      # cat("Doc sim finished.\n")
      lsaDocSim.out
    }
  }, ignoreNULL = FALSE)
```

- This is the block of code that creates the topic model. It is an `eventReactive` on the button `input$submit_button`. `ti` holds the course code you want similar courses for and `sg` is what year level you want the suggested courses to be in.
- Next the code checks if `ti` and `sg` are null and if not `use_or_create_env()` is called. The code for this function is in `functions.R` and it holds the logic for creating a Python environment using the `reticulate` package.
- Data is then read in with course descriptions and a Latent Semantic Analysis topic model is trained using `lsaDocSim(ti, sg, df)`. Code can be found in `functions.R`. This function loads all the necessary Python libraries, cleans all the text using common NLP preprocessing techniques, and fits the model. Then, document

embeddings are retrieved and the cosine similarity between the course in `ti` and all other courses is calculated. Finally, the top 3 most similar courses are returned.

```
# Render the course recommendation UI
output$courseRecSim <- renderUI({
  cat("Loading course suggestion data for display.\n")
  data <- reactive_data()
  cat("Data loaded.\n")
  if (!is.null(data)) {
    tagList(
      tableOutput("table_view"),
      checkboxInput(
        "checkbox1rec",
        paste(data$Course.Name[1], data$Course.Code[1]),
        value = FALSE
      ),
      checkboxInput(
        "checkbox2rec",
        paste(data$Course.Name[2], data$Course.Code[2]),
        value = FALSE
      ),
      checkboxInput(
        "checkbox3rec",
        paste(data$Course.Name[3], data$Course.Code[3]),
        value = FALSE
      )
    )
  }
})

# Dataframe to otuput for course sugesstion topic model
output$table_view <- renderTable({
  reactive_data()
})
```

- This is the block of code responsible for rendering the `uiOutput` with id `courseRecSim`. `reactive_data()` creates and retrieves the topic model output. Then it creates 3 checkboxes, one each of the 3 similar courses returned.
- Then in the next code block, a table of the courses and their similarity scores is rendered using `renderTable({...})`

```
 # Logic to handle adding suggested courses from topic model
observeEvent(input$checkbox1rec, {
  data <- reactive_data()
  course_name <- data$Course.Name[1]
  course_code <- data$Course.Code[1]
  year <- data$Course.Code |> substr(1, 1) |> as.numeric()
```

```
    cat("This is what wants to be added and deleted from the database thanks to suggestions
1\n")
    cat("\n")
    cat(paste(course_name, course_code))
    cat("\n")
    cat(course_code)
    cat("\n")
    cat(year[1])
    cat("\n")

    if (input$checkbox1rec) {
      # Insert the course into the database if checked
      query <-
        sprintf(
          "INSERT INTO selected_courses (course_name, course_code, year) VALUES ('%s', '%s',
%d)",
          paste(course_name, course_code),
          course_code,
          year[1]
        )
      dbExecute(db, query)
    } else {
      # Remove the course from the database if unchecked
      query <-
        sprintf(
          "DELETE FROM selected_courses WHERE course_name = '%s' AND course_code = '%s' AND year
= %d",
          paste(course_name, course_code),
          course_code,
          year[1]
        )
      dbExecute(db, query)
    }
  })

  observeEvent(input$checkbox2rec, {
    data <- reactive_data()
    course_name <- data$Course.Name[2]
    course_code <- data$Course.Code[2]
    year <- data$Course.Code |> substr(1, 1) |> as.numeric()

    cat("This is what wants to be added and deleted from the database thanks to suggestions
2\n")
    cat("\n")
    cat(paste(course_name, course_code))
    cat("\n")
    cat(course_code)
    cat("\n")
    cat(year[1])
    cat("\n")
```

```
    if (input$checkbox2rec) {
      # Insert the course into the database if checked
      query <-
        sprintf(
          "INSERT INTO selected_courses (course_name, course_code, year) VALUES ('%s', '%s',
%d)",
          paste(course_name, course_code),
          course_code,
          year[1]
        )
      dbExecute(db, query)
    } else {
      # Remove the course from the database if unchecked
      query <-
        sprintf(
          "DELETE FROM selected_courses WHERE course_name = '%s' AND course_code = '%s' AND year
= %d",
          paste(course_name, course_code),
          course_code,
          year[1]
        )
      dbExecute(db, query)
    }
  })

  observeEvent(input$checkbox3rec, {
    data <- reactive_data()
    course_name <- data$Course.Name[3]
    course_code <- data$Course.Code[3]
    year <- data$Course.Code |> substr(1, 1) |> as.numeric()

    cat("This is what wants to be added and deleted from the database thanks to suggestions
1\n")
    cat("\n")
    cat(paste(course_name, course_code))
    cat("\n")
    cat(course_code)
    cat("\n")
    cat(year[1])
    cat("\n")

    if (input$checkbox3rec) {
      # Insert the course into the database if checked
      query <-
        sprintf(
          "INSERT INTO selected_courses (course_name, course_code, year) VALUES ('%s', '%s',
%d)",
          paste(course_name, course_code),
          course_code,
```

```
          year[1]
        )
      dbExecute(db, query)
    } else {
      # Remove the course from the database if unchecked
      query <-
        sprintf(
          "DELETE FROM selected_courses WHERE course_name = '%s' AND course_code = '%s' AND year
= %d",
          paste(course_name, course_code),
          course_code,
          year[1]
        )
      dbExecute(db, query)
    }
  })
```

- This block of code is responsible for adding checked suggested course boxes to the database of selected courses. This will then also render the courses on the graph. Each created checkbox has an `observeEvent` on it such that when it is checked, the course is added to the database.