Supplementary Material for "Interactively visualizing distributional regression models with distreg.vis"

Stanislaus Stadlmann
and
Thomas Kneib
Chairs of Statistics and Econometrics,
Georg-August-Universität Göttingen, Germany

March 15, 2021

A Implementation of `distreg.vis`

Understanding the inner workings of `distreg.vis` is best done by starting at the two most important functions, `plot_dist()` and `plot_moments()`. Each function is a direct answer to the two central questions which led to the creation of `distreg.vis`:

1. What exactly does my predicted distribution look like, based on selected covariate combinations?

2. How do the expected moments of my target distribution vary over the range of a specific variable of interest?

The remaining functions of the `distreg.vis` solely feature a supportive character easing the overall computative process. Here, we start with `plot_dist()`:

A.1 Visualization of distribution predictions

To accurately display the predicted distribution of a covariate combination, the first step is to gather the appropriate distributional parameters from which to build the probability density (pdf) or cumulative distribution function (cdf). These parameters are obtained by choosing certain covariate combinations and then using the estimated model for prediction. These covariate combinations are usually user-picked, but `distreg.vis` offers a few tools to ease the choosing process, as shown below.

A.1.1 Choosing covariate combinations

This Section introduces tools to make using both `plot_moments()` and `plot_dist()` easier by creating datasets to predict with in the correct format. To quickly obtain a `data.frame` object with all variables set to their mean, one can use `model_data()` to obtain the covariates with which the model was estimated, and then `set_mean()` to calculate the average values of each covariate. Both functions and their arguments are repeatedly used in and as such are part of the main functions in `distreg.vis`, `plot_dist()` and `plot_moments()`.

The usage of `model_data()` is as follows:

```
model_data(model, dep = FALSE, varname = NULL)
```

where `model` represents one of `distreg.vis`' supported model classes. It returns a `data.frame` object containing the explanatory variables, except when `dep = TRUE` (returns vector of dependent variable) or when `varname` is specified in character form (returns a vector of a specific named explanatory variable).

After obtaining the explanatory variables, reducing its dimensions to the mean is achieved with `set_mean()`, the usage of which is as follows:

```
set_mean(input, vary_by = NULL)
```

With the argument `input`, a `data.frame` object is specified, which can conveniently be the output of a `model_data()` call. Then, average values are calculated of each column in `input`. For categorical variables, the first level of the underlying factor class is taken, which in model estimation typically corresponds to the reference category.

Using the argument `vary_by()`, the user can specify an explanatory variable in character form, over which the returned `data.frame` is "varied". This means that the returned object consists of the same number of columns as before, but with multiple rows and varying values in the covariate of choice. If this variable is categorical, then the varying values consist of the possible categories. In numeric cases, a sequence of five values, ranging from the 2.5% to the 97.5% quantile is created. All other variables stay constant at their mean or reference category. This argument is very useful for a display of marginal distributions, based on the resulting `data.frame` object.

To show an example of the ease of use of the aforementioned functions, we consider the same model that was fit in Section 2 of the main paper. To retrieve the explanatory variables of that model, and set them to the mean with varying values in the categorical covariate `education`, we can run the following code:

```
R> df <- set_mean(input = model_data(wage_model), vary_by = "education")
R> row.names(df) <- levels(Wage$education)
R> df
                  ethnicity year          education age
1. < HS Grad        1. White 2006       1. < HS Grad  42
```

```
2. HS Grad          1. White 2006         2. HS Grad  42
3. Some College     1. White 2006    3. Some College  42
4. College Grad     1. White 2006    4. College Grad  42
5. Advanced Degree  1. White 2006 5. Advanced Degree  42
```

From the results of the above code chunk, we can now quickly make parameter predictions and obtain either graphs with the marginal distribution (`plot_moments()`), or marginal influence plots `plot_moments()`. Further defining the `row.names` of the `data.frame` to be the different education levels ensures improved legends in further graphs.

A.1.2 Parameter predictions

Predicted parameters of the target distribution are usually outputs of `predict.object()` functions for each regression model class. This is no different in the case of `bamlss`, `gamlss` or `betareg`, for which those functions also exist. Unfortunately, the usage of all three `predict` functions is not consistent over its respective packages, and sometimes not even within the package throughout parameters of different target distributions.

For example, `predict.bamlss()` returns a vector if the predicted distribution only consists of one parameter, a list of vectors if it consists of more than one, and a list of matrices if the MCMC samples of the predicted parameters are desired. To ensure unity and guarantee type consistency over the supported packages outcome classes, `preds()` was written. Without worrying about class-specific function arguments, it offers a consistent way of obtaining predictions based on specific covariate combinations. Its usage is as follows:

```
preds(model, newdata = NULL, what = "mean", vary_by = NULL)
```

where `model` represents a fitted `gamlss`, `bamlss` or `betareg` object and `newdata` a `data.frame` with different covariate combinations in each row. If `newdata` is omitted (`= NULL`), then the mean of the explanatory variables are used for prediction, utilizing `set_mean()`. This can be used in combination with the argument `vary_by`, which then creates a `newdata` that consists of varying values in one specific variable (see Chapter A.1.1 for details).

The argument `what` specifies whether the predicted parameters should be directly calculated (`what = "mean"`), or the output should consist of samples of the predicted parameter (`what = "samples"`). This option is only available for the `bamlss` model class, and is necessary for exact estimates of the predicted moments or other measures derived from the parameters and for the construction of credible intervals at a later stage (e.g., for plotting).

Obtaining samples for the predicted parameters of our target distribution, based on the different education levels defined in `df` can now be done as follows, using the option `what = "samples"`:

```
R> pp_samples <- preds(model = wage_model,
+                      newdata = df,
+                      what = "samples")
R> lapply(pp_samples, head, 2)


$`1. < HS Grad`
          mu     sigma
[1,] 4.506987 0.2850489
[2,] 4.485811 0.2498074

$`2. HS Grad`
          mu     sigma
[1,] 4.630472 0.2872791
[2,] 4.592440 0.2883318

$`3. Some College`
          mu     sigma
[1,] 4.759729 0.2878243
[2,] 4.712548 0.2792889

$`4. College Grad`
          mu     sigma
[1,] 4.864325 0.3270728
[2,] 4.816144 0.3139536
```

```
$`5. Advanced Degree`
          mu     sigma
[1,] 4.999823 0.3329738
[2,] 4.991542 0.3368283
```

In this case, the object `pp_samples` was produced as a `list` with as many elements as preselected covariate combinations. Each element is a matrix containing samples in rows (by default `bamlss` keeps $n = 1001$, of which the first two rows are shown above) for all distributional parameters (in this case two: μ, σ). The two plotting functions of `distreg.vis` automatically recognize the object and can extract information from both Maximum-Likelihood estimates (`gamlss` or `betareg`) or MCMC samples (`bamlss`). If the predicted parameters are provided in the form of samples (as a `list` object in `R`), `plot_moments()` is able to display credible intervals above and below the predicted moments.

A.1.3 Visualize predicted distributions

To get a feel for the predicted distributions and their differences, it is best to visualize them. In combination with the obtained parameters from `preds()`, the function `plot_dist()` finds the necessary distribution functions (probability density function or cumulative distribution function) from the respective packages and then displays them graphically.

Creating the distributional graph is a five step process, beginning with the right input, the arguments. The usage of `plot_dist()` is as follows:

```
plot_dist(model, pred_params = NULL, palette = "viridis",
 type = "pdf", rug = FALSE, vary_by = NULL, newdata = NULL)
```

As before, `model` depicts the fitted distributional regression model. The argument `pred_params` takes the previously predicted parameters of the fitted distribution. Options `palette` and `rug` yield aesthetic changes; the first one is able to process any character string that resembles either `"default"`, which results in displaying the default color palette of `ggplot2` (Wickham, 2016), `"viridis"` for a colorblind-friendly palette from the `viridis` package (Garnier, 2017) or a palette from the `RColorBrewer` package (Neuwirth, 2014).

If the user wishes to leave out the step of specifying predicted parameters with `pred_params`,

the argument `newdata` can be provided instead. This argument will then be passed onto `preds`, which internally computes the predicted parameters used for plotting the distribution functions. If neither `pred_params` nor `newdata` are specified, then `plot_dist()` uses the mean values of the explanatory variables with `set_mean()` to predict the parameters used for plotting. The argument `vary_by` is also passed on to `set_mean()` which, if specified, leads to `plot_dist()` displaying marginal distributions over the range or categories of a specified variable (see Section A.1.1 for details).

The last argument `rug`, as the name implies, adds a "rug graph" below the density/cumulative distribution graphs, which shows small vertical strips indicating where actual observations of the dependent variable lie. Using the argument `type`, one can switch between probability density functions (`"pdf"`) or cumulative distribution functions (`"cdf"`).

After `plot_dist()` has received all necessary arguments, it executes validity checks to ensure the argument's correct specification. This includes controlling for the correct `model` class, checking whether the distributional family can be used safely and whether cdf or pdf functions for the modeled distribution are present and ready to be graphically displayed. If this is the case, the internal `fam_fun_getter()` is used to create a `list` with two functions pointing to the correct pdf and cdf functions for the distributions from either the `gamlss`, `bamlss` or `betareg` namespace.

To graphically display the previously obtained functions, the only remaining unanswered question is one about the limits: What should be the axis' ranges? To accomodate this task, the internal function `limits()` was created. The default option in `distreg.vis` for the x-axis on both cdf and pdf function displays are the 0.1% and the 99.9% quantiles of the predicted distribution. If multiple distributions are displayed, always the minimum and maximum values of all calculated quantiles are used.

Both the validity checks and the `limits()` function make use of specific information about every available distribution in the supported distributional regression packages. This information is stored in a `data.frame` object called `dists` with the following form:

```
R> str(dists)
```

```
'data.frame': 125 obs. of  8 variables:
$ dist_name  : chr  "BB" "BCCG" "BCCGo" "BCPE" ...
$ class      : chr  "gamlss" "gamlss" "gamlss" "gamlss" ...
$ implemented: logi  FALSE TRUE TRUE TRUE TRUE TRUE ...
$ moment_funs: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
$ type_limits: chr  "both_limits" "one_limit" "one_limit" "one_limit" ...
$ l_limit    : int  0 0 0 0 0 0 0 0 0 0 ...
$ u_limit    : int  10 NA NA NA NA NA NA 1 1 1 ...
$ type       : chr  "Discrete" "Continuous" "Continuous" "Continuous" ...
```

The `dists` object contains one row for each distribution, and columns with the following content:

- `dist_name`: Name of the distribution.
- `class`: Either `"bamlss"`, `"gamlss"` or `"betareg"` detailing from which package the target distribution comes from.
- `implemented`: Is this distribution generally usable for `plot_dist()`, and was this usage already tested?
- `moment_funs`: Are functions implemented with which to calculate the moments of the distribution, given the parameters? This column is notably relevant for `plot_moments()`, in which the predicted moments are displayed.
- `type_limits`: Details the range the values from the distribution can have. Can be `"both_limits"`, `"one_limit"`, `"no_limit"` and `"cat_limit"` (for categorical distributions).
- `l_limit, u_limit`: Integers detailing where the limits of the distributions lie.
- `type`: Character string for the type of distribution. Can be "Discrete", "Continuous", "Mixed" and "Categorical".

Following a successful calculation of the plot limits, the graph itself can be created. Internally, `distreg.vis` divides between continuous, discrete and categorical distributions.

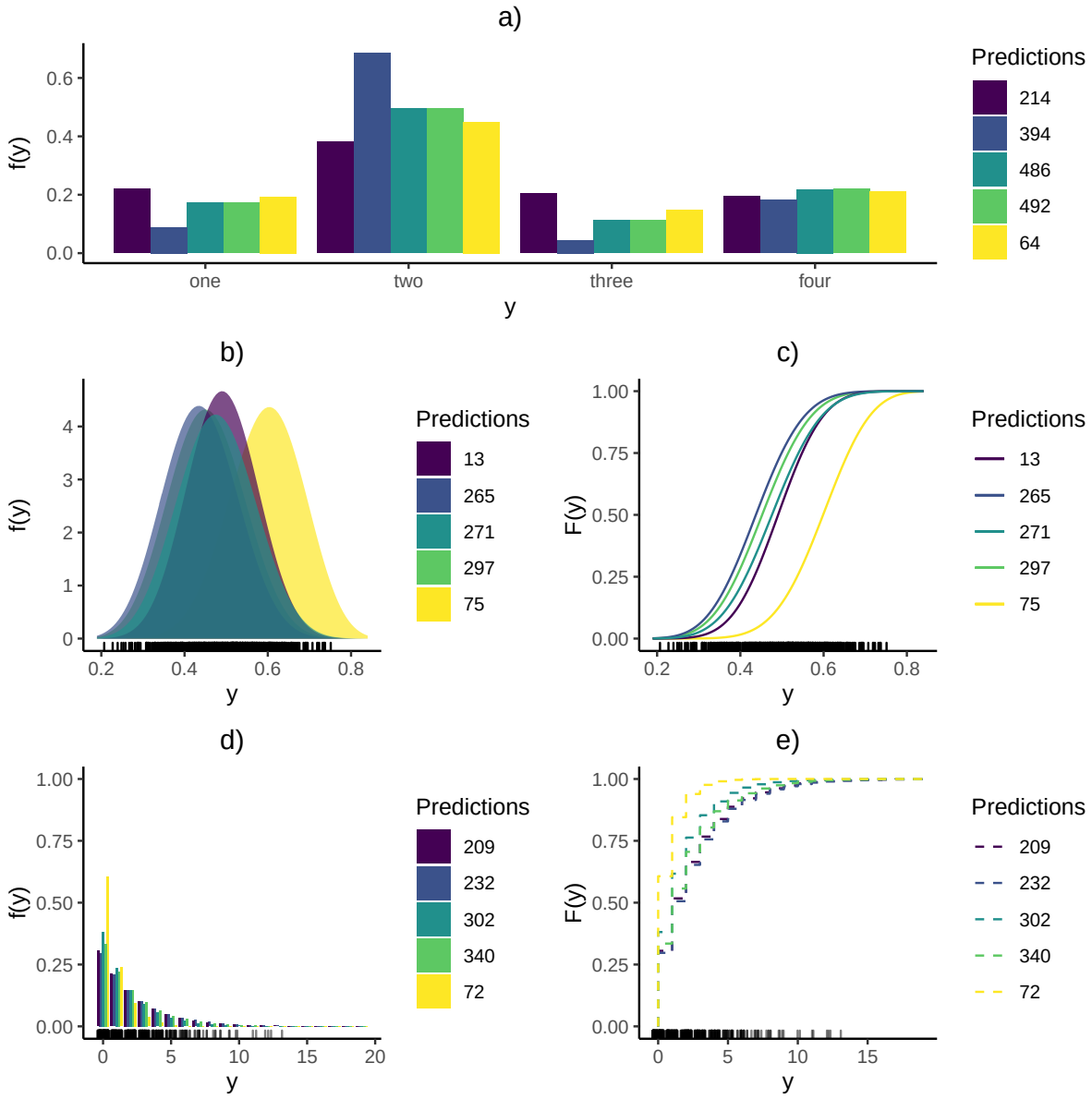

Figure A1: Plot outcome possibilities in `plot_dist()` displayed with five different predictions in five examples: a) a multinomial probability density function, b) an expected pdf with the respective cdf of a Beta distribution shown in c), and the expected pdf and cdf pair of a geometric distribution in d) and e), respectively.

Continuous distributions are displayed as filled line plots, while discrete and categorical distributions take bar graph shapes.

For plotting, `distreg.vis` relies on the `ggplot2` package (Wickham, 2016). After an empty graph is constructed, the previously obtained cdf or pdf functions are evaluated for each predicted parameter combination and all values inside the calculated plot limits.

Figure A1 lays out examples of the five different shapes of `plot_dist()`: The probability density function (pdf) of continuous, discrete and categorical distributions, as well as the cumulative distribution function (cdf) of continuous and discrete distributions. In each of those cases (subplots a) to e)), a dataset was simulated of an appropriately chosen target distribution, which was then used for model estimation. Based on these models and five arbitrary covariate combinations from the datasets, the predicted pdf's and cdf's were computed and displayed.

A.2 Visualization of effect influence

The target of the second main function, `plot_moments()`, is to display the influence of a selected effect on the predicted moments of the modeled distribution. The motivation for computing influences on the moments of a distribution is its interpretability: In most cases, the parameters of a distribution do not equate the moments and as such are only indirectly location, scale or shape properties, making the computed effects hard to understand.

When `plot_moments()` is called, the provided function arguments are evaluated first. They are specified as follows:

```
plot_moments(model, int_var, pred_data = NULL, rug = FALSE,
 samples = FALSE, uncertainty = FALSE, ex_fun = NULL,
 palette = "viridis", vary_by = NULL)
```

Here, `model` represents the fitted distributional regression object (as before). The argument `int_var` stands for "variable of interest" and consists of a character string with the name of the explanatory variable, whose effect influence should be plotted. This covariate can be either categorical or continuous. The argument `pred_data` is specified using a `data.frame` object that consists of the values that the remaining variables should attain, while `int_var` varies. If this `data.frame` has multiple rows, the influence plot is re-drawn for each different covariate combination. This is a neat way to, for example, visualize interaction effects between two variables.

If `pred_data` is not specified (left at `NULL`), `plot_moments()` will utilize `set_mean()` in combination with `model_data()` to use mean values of the explanatory variables for

prediction. If the argument `vary_by` is specified, it will be passed onto `set_mean()` for creating a `data.frame` object in which one variable has varying values, while all other stay at their mean (see Chapter A.1.1 for details).

Using the arguments `samples` and `uncertainty`, one can exploit the advantages that Bayesian regression models have to offer. With `samples = TRUE` the predicted moments are computed by transforming the samples of the predicted parameters to the moments, and then calculating the mean of the samples over the range of `int_var`. If the argument `samples` was chosen to be `FALSE` (the default case), then the mean of the samples is taken before they are transformed to the predicted moments.

This distinction is especially relevant in nonlinear parameter-to-moment transformations, where taking the average before transforming the samples yields inaccurate results. Setting `samples = FALSE` should therefore only be used in situations where one is interested in quick and rough effect influences. For complete accuracy setting `samples = TRUE` is preferred.

With `uncertainty`, the user can construct credible intervals for the influence plots. If this argument is set to `TRUE`, then empirical 2.5% and 97.5% quantiles are computed for the predicted moments of the entire range of our variable of interest, `int_var`. Using this method, it is possible to detect complex significant differences between the moments of a categorical variable, for example. Both `samples` and `uncertainty` can only be set to `TRUE` for `bamlss` model classes.

The argument `ex_fun` takes a character string with the name of an R function that currently exists in the user's global environment. This function can be any function which takes the predicted parameters as input and yields a single number as an output. For situations where metrics depend on the entire predicted distribution of the target variable, like the Gini coefficient (Lerman and Yitzhaki, 1984), this is especially useful. Combined with the sample-based approach of `bamlss`, we can even obtain uncertainty measures for those user-defined metrics.

After typical argument validity checks, a sequence is created that covers the entire range of the variable of interest, `int_var`. This sequence is then combined with `pred_data` to

obtain a `data.frame` object where the main variable varies but the previously specified other variables stay constant. The newly created `data.frame` is further passed to the `preds()` function, which computes predicted parameters for every row.

Depending on whether `samples` was set to `TRUE` or not, the output of the called `preds()` function is now either a `data.frame` with the same dimensions as `pred_data` and with the parameters replaced by the predicted moments, or a `list` with one element for each user-specified covariate combination consisting of the parameter samples which were transformed to the expected moments of the target distribution. Afterwards, the mean of all moment samples at each point of `int_var`'s range is calculated, resulting in all necessary values to create the resulting influence graph. If `uncertainty` is also `TRUE`, the upper and lower limits of the credible intervals are constructed.

Obtaining the predicted moments and possibly lower and upper bounds of the credible intervals means that the resulting graph can then be created. To illustrate the potential of `plot_moments()`, we will again focus on the introductory example, where the question of interest is the relationship between annual income and different socioeconomic variables. Calling `plot_moments()` for both a numeric (`age`) and a categorical variable (`ethnicity`) can be done as follows:

```
R> gini <- function(par) {
+    2 * pnorm((par[["sigma"]] / 2) * sqrt(2)) - 1
+  }
R> plot_moments(
+    wage_model,
+    int_var = "age",
+    pred_data = df_new,
+    samples = TRUE,
+    uncertainty = TRUE,
+    ex_fun = "gini",
+    rug = TRUE
+  )
R> plot_moments(
+    wage_model,
+    int_var = "ethnicity",
```

```
+     pred_data = df_new,
+     samples = TRUE,
+     uncertainty = TRUE,
+     ex_fun = "gini"
+   )
```

Figure A2 shows the outcome of the above code, which represents two of the three different graph outcomes that `plot_moments()` can have. Part a) focuses on the relationship between the `age` of observed individuals, a numeric variable, and the first two expected moments of the modeled income distribution (leftmost and middle graph). The different lines represent multiple education levels, which were specified in the `data.frame` object called `df` (p. 3). For a less cluttered visualization, this portrayal of the `age` influence features only three different education levels resulting in three different lines.

Even though the first two expected moments are already informative, one might be interested in other specific measures being dependent on the parameters of a distribution. This could include higher-order moments like skewness or more specific measures, like income inequality. Previously, finding out how this measure varies over a variable would involve many lines of code. With `distreg.vis`, specifying an external function is easy. The very right graph of part a) portrays the external function (`gini`) that was specified with the argument `ex_fun`. It represents the Gini coefficient of the distribution, which is a measure of inequality of a distribution (Lerman and Yitzhaki, 1984). Similarly to the expected moments, we also construct credible intervals around the self-specified function `gini`, as shown in Figure A2.

Below and above the lines of the varying moment and external function levels lie shaded areas, which represent the uncertainty of the influence. These are credible intervals, i.e. empirical 2.5% and 97.5% quantiles of the obtained MCMC samples transformed to be expected moments. The advantages of having uncertainty areas are numerous, but include the direct visibility of significant differences between two different categories chosen with the `data.frame` object called `pred_data`. For example, the left plot of part a) shows a significant difference in the expected income for all three chosen education levels, because the credible intervals do not overlap in almost the entire range of `age`.

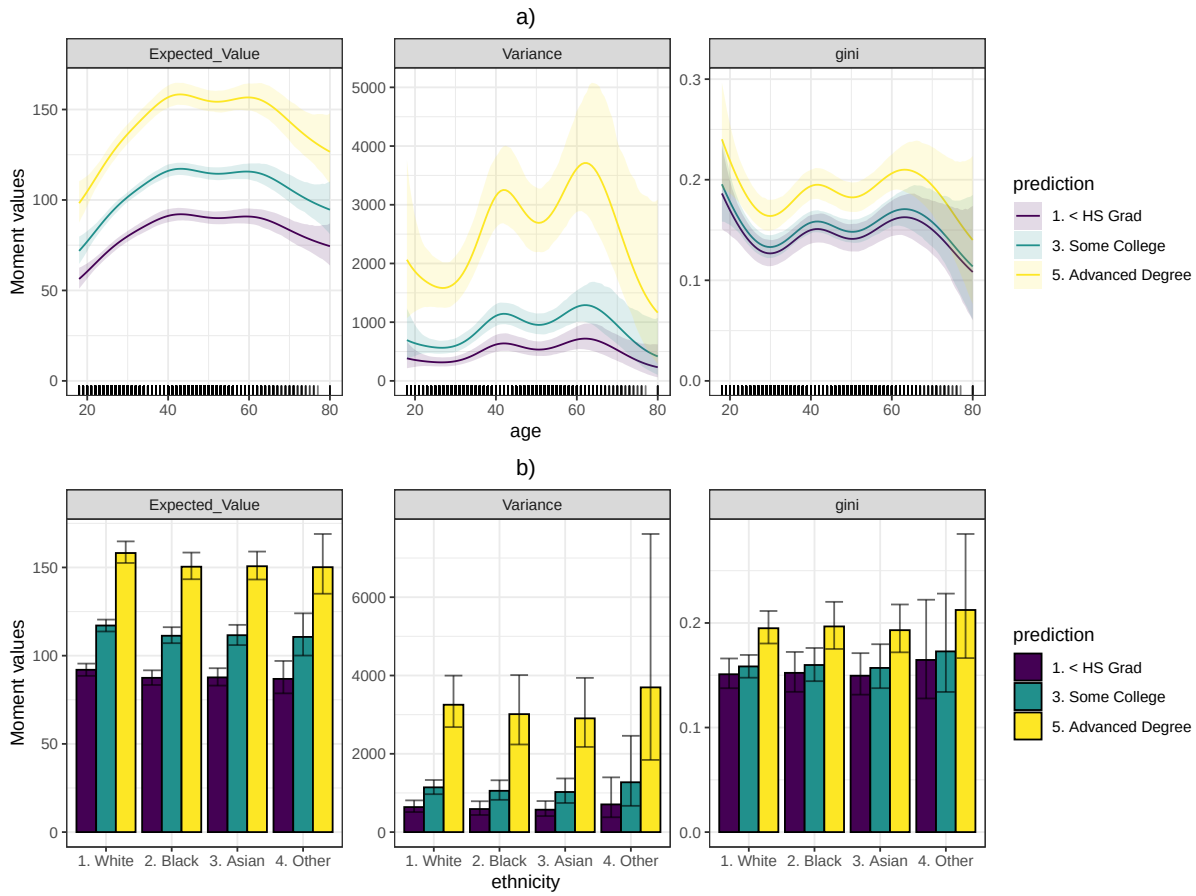

Figure A2: Plot outcome possibilities in `plot_moments()`: Part a) shows the expected moments (`Expected_Value` and `Variance`) and a user-specified function (`gini`) over the range of a continuous covariate, including its credible intervals. Part b) displays the expected moments and the external function over the range of a categorical variable, `ethnicity`.

The bar graphs of Figure A2 part b) appear if a categorical variable is chosen as the variable of interest `int_var`. In this case it was chosen to be the `ethnicity` of individuals. The x-axis portrays all different categories that the variable can attain. Then, the height of each bar features the expected mean or variance of the predicted distribution when `int_var` reaches the specific category. The different bars in each category represent again the covariate combinations specified in the `pred_data` argument. Due to the `pred_data` covariate values only varying in the `education` levels of the individuals, one could understand part b) of Figure A2 as the predicted expected value, variance and external function of the modeled distribution, broken down to each combination of both `education` and `ethnicity`.

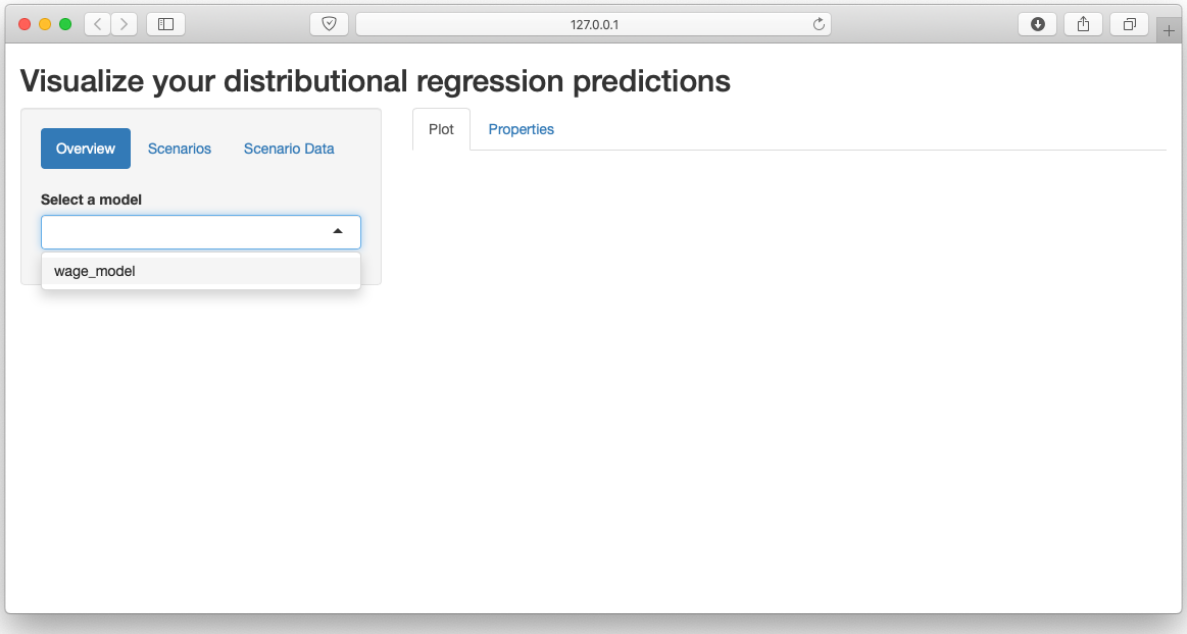

Figure B3: Layout of `distreg.vis` after starting the application.

Similar to the continuous case, an external function can also be included for discrete variables of interest. The function called `gini` was used again, calculating the Gini coefficient for the log-normal distribution. The small error bars, also only available for `bamlss` objects, show the credible intervals around the predicted moments in each category. The significance between categories can now be checked in two ways - between the categories of `int_var` and between the different covariate combinations chosen in `pred_data`. In this case, we can see a significant wage difference between the `education`, but not the `ethnicity` levels.

B The Graphical User Interface

After executing `vis()`, a new browser window with the started application is opened up. Figure B3 shows the layout of the application, which is then displayed in the user's browser.

As visible in Figure B3, the layout of `distreg.vis`' GUI is divided into two segments, which have their own tabs the user can click on. In each segment, one of those tabs is always displayed. The left segment, with tabs "Overview", "Scenarios" and "Scenario Data", is concerned with model-related settings. The right segment, with tabs "Plot" and

"Properties" is used to display graphs and properties in reaction to user inputs on the left segment.

B.1 Overview Tab

The purpose of the overview tab is to display descriptive details about fitted distributional regression models. After the GUI of `distreg.vis` is started up, it solely consists of a drop-down list where the user can select the model on which the further analysis is to be based. Entries in this list are created by the internal function `search_distreg()` which searches the working directory of the current user for any object of the classes `bamlss`, `gamlss`, `betareg` or `betatree` (also from the `betareg` package) . Figure B3 shows only one entry, `wage_model`, representing the model fitted with the code provided in Section 2 of the main paper.

After a model is selected, the overview tab expands to show an outline of the fitted `bamlss` model. Specifically, as shown in Figure B4, the tab displays two parts, called "Model Family" and "Model Equations". "Model Family" shows the family of the model's target distribution, as well as the parameters which can be modeled including their link functions. In the case of `wage_model`, the family "lognormal" with parameters μ and σ and link functions "identity" and "log" can be obtained. "Model Equations" displays the way covariate effects were specified. We can confirm that for `wage_model`, the effect of age on both μ and σ is specified using a smooth spline.

B.2 Scenarios Tab

In this tab, the user can interactively specify covariate values for each explanatory variable, thereby creating a "Scenario". After these combinations are finalized by clicking a button, the predicted distribution based on the previously selected model is then graphically displayed. This can be repeated with different covariate values to compare the predictions for multiple covariate combinations.

As displayed in Figure B5, the top of the tab consists of two buttons, "Create Scenario" and "Clear Scenarios". Right below, web widgets for each explanatory variable

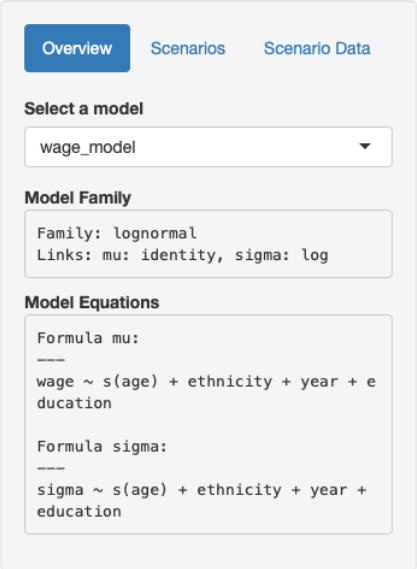

Figure B4: Expanded overview tab after model selection.

are visible. Here, `distreg.vis` executes a check for the type of each explanatory variable and then constructs different web application elements depending on that information. Categorical covariates receive selector boxes (R function `shiny::selectInput()`) with the variable's possible categories, while for numeric variables slider modules are created (`shiny::numericInput()`), ranging from the variables minimum to maximum value. The default value for numeric covariates is its arithmetic average, while for categorical covariates the first level of the resulting `factor` variable is used.

To add a new scenario, the user can now specify a value for each variable and click on "Create Scenario". This triggers the plotting window, which in turn displays the predicted distribution for the specified covariate combination. To compare two or more scenarios, the user can change the specified covariate values and again press "Create Scenario". This way, the graph window will display predicted pdf/cdf functions for each specified covariate combination. Deleting all previously specified combinations can be achieved by clicking "Delete Scenario".

Beneath the visual components of the "scenarios" rests a small database, which stores each covariate combination the user has specified. This database, created with `shiny::reactiveValues()` forms the basis for all other tabs which analyze those covariate com-

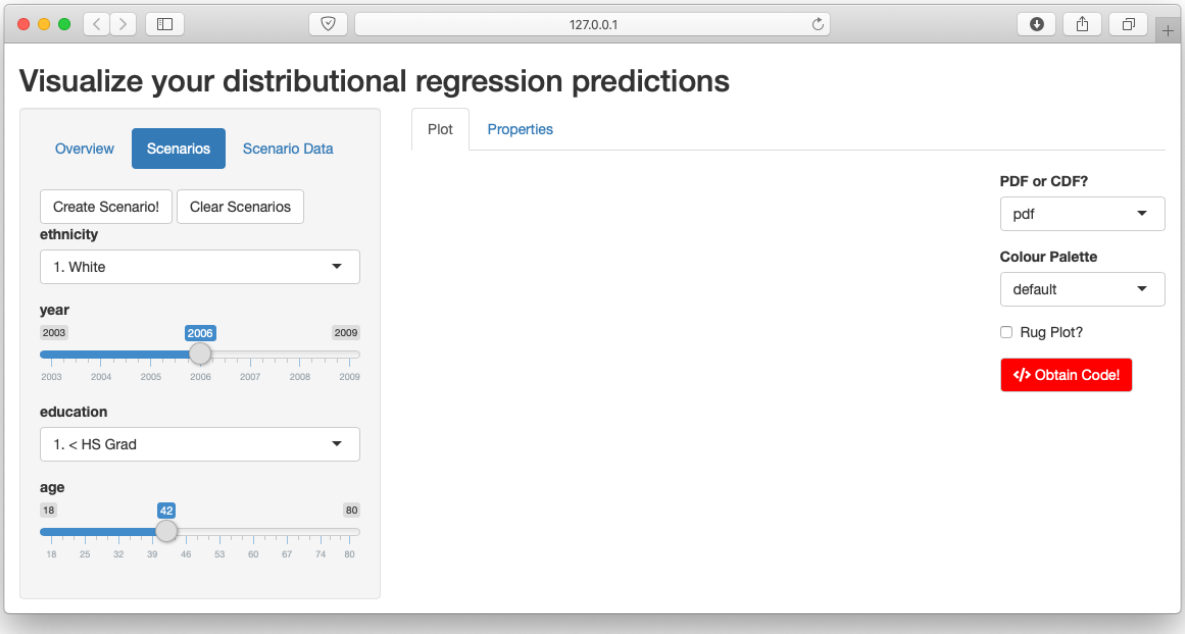

Figure B5: "Scenarios" tab of the `distreg.vis` GUI.

binations. It has a "reactive" nature, such that all tabs depending on it are automatically updated when a database value changes.

We can now easily create a graph for the predicted distribution of wages depending on every `education` level. To do so, we set the covariates to the following values:

- `ethnicity: 1. White`
- `year: 2006`
- `education: 1. < HS grad`
- `age:` 42 $(= \overline{age})$

Then, the "Create Scenario" Button is pressed. This is done four more times, with each time seeing a rise in education level by one category. Thus, we can now view the according wage distributions for a 42-year-old white male across all levels of education (Figure B6). Furthermore, the "rug plot" option was selected.

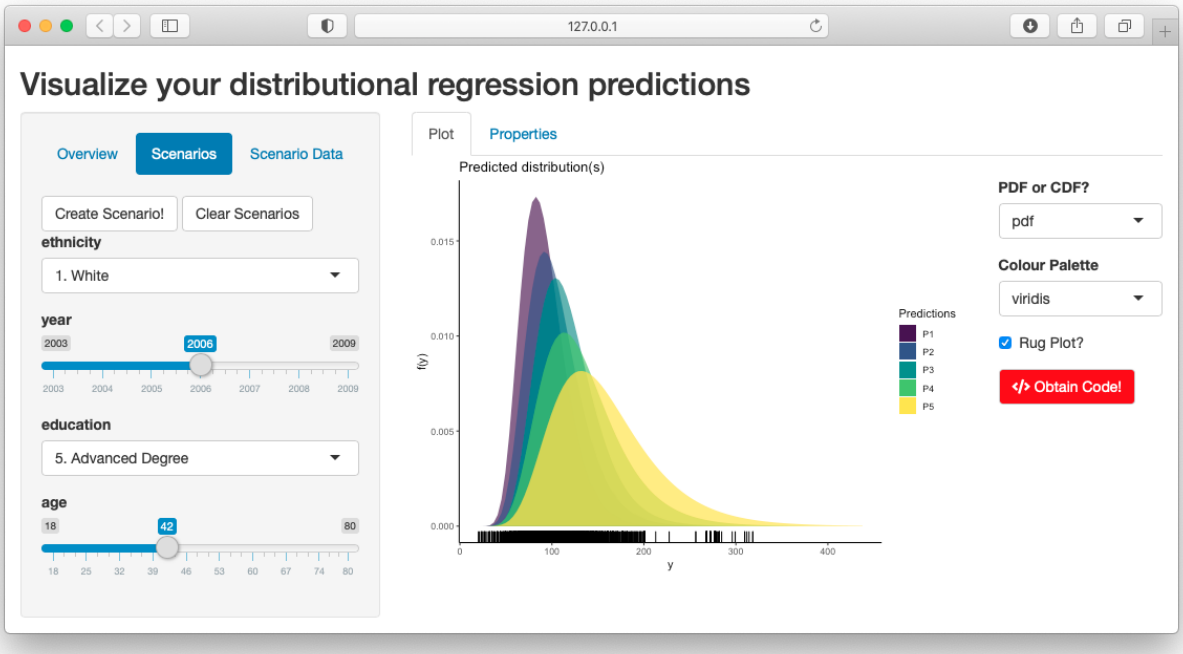

Figure B6: Plot tab output when specifying five different scenarios with different education levels.

B.3 Plot Tab

The "Plot" tab, which is located in the right-hand segment of `distreg.vis`, reacts to user interaction in the Scenarios tab. Every time the "Create Scenarios" button is pressed, the tab is updated. Specifically, the data which the user inputs in the left tab is passed onto `distreg.vis::preds()`, which then computes predictions for the response distribution parameters. Afterwards, the predicted parameters are inserted into the probability density or the cumulative distribution function, which is then graphically displayed. This procedure is repeated for each covariate combination.

Figure B6 shows the plot output for five different scenarios based on the `Wage` dataset. Noticeable in Figure B6 to the right side of the plot are multiple web elements for user interaction. The first element, found below the description "PDF or CDF?", provides the ability to switch between displaying the probability density function (pdf, the default value) or the cumulative distribution function (cdf). Figure D11 in Section D shows Figure B6 after the "cdf" option was selected.

The second element gives the option to select a different color palette. Its default value is "default", which uses the built-in color palette provided in `ggplot2` (Wickham,

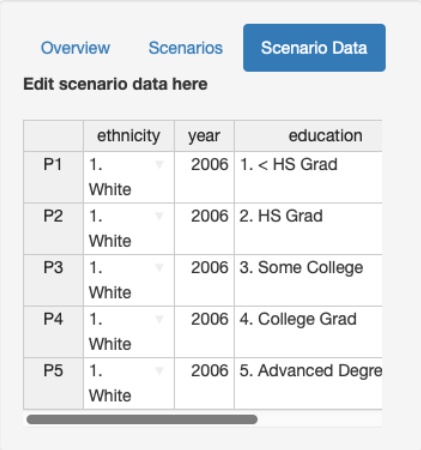

Figure B7: "Scenario Data" tab.

2016). The selected color palette in Figure B6 is "viridis" (from R package `viridis`), a colorblind-friendly palette (Garnier, 2017).

Using the third element labelled "Rug plot?", a small strip chart at the bottom of the graph can be added, displaying the observation frequency at each level of the target variable's range. The fourth web element on the right side of the plot output, a red button with the description "Obtain Code!", adds reproducibility to the plot. When clicked, a modal window pops up with R commands that, if executed in the main R console with the user's current working environment, directly recreates the graph currently being displayed in the "Plot" tab. Figure D12 in Section D shows the modal window which arises when pressing the "Obtain Code!" button in the interface of Figure B6.

B.4 Scenario Data Tab

While the "Scenario" tab gives the ability to quickly specify covariate values, sometimes the user might want to type in exact values on which to make predictions. Furthermore, one might want to see what values were previously specified in the "Scenarios" tab. For both reasons the tab "Scenario Data" was created in `distreg.vis`' left segment.

Figure B7 shows the tab's layout. As visible, the only present web element is a table placed underneath the description "Edit scenario data here". This table, created with the R package `rhandsontable` (Owen, 2016) represents the editable version of all data input from the "Scenarios" tab. Columns represent the specified covariates, while one row counts

as one "scenario". In Figure B7, it is possible to see that only three columns of the original six are currently visible. This cut-off was integrated to prevent an overlap with the plot. Nevertheless, the user can use the scrolling bar at the bottom to reach other columns. One additional column is added to the existing covariates, labelled "rownames". With this column, each "scenario" obtains a new label, which then automatically transfers to the legend of each graph.

To edit an observation from categorical variables, users can click on a small drop-down button in the cell which can be edited. The table recognizes categorical variables and will then provide a menu where the desired value is to be selected. With numeric variables, users can select the cell and then input the value they wish to make predictions with. Values in logical variables can be specified by checking or un-checking a box in the cell.

With the ability to specify own covariate values also comes the ability in numeric cases to type in numbers that are not in the original variable's range. In the case of `cnorm_model`, this could mean that one tries to make predictions for incomes in the year `2050`, which is far beyond $max(year) = 2009$. To circumvent the irresponsible usage of model predictions, `distreg.vis` will display a warning pop-up message with the name of the covariate combination (`P` plus the respective number if not changed with column "rownames") where values were specified which are out of the original variable's range (Section D: Figure D13).

B.5 Properties Tab

Previous chapters have described that the "Plot" tab visualizes the predicted distributions for each covariate combination specified in the "Properties" tab. However, while differences in distributions for each combination were visible (e.g., in Figure B6), it is only indirectly possible to infer influences of the complete range of a numerical covariate on the distributional moments.

To provide this functionality, the tab "Properties" was implemented in `distreg.vis`, located in the right-hand segment and greatly building upon the function `plot_moments()`. When opened, the "Properties" tab reveals two sub-tabs: "Influence graph" and "Table".

As visible in Figure B8, the "Influence graph" tab consists of a graph on the left side

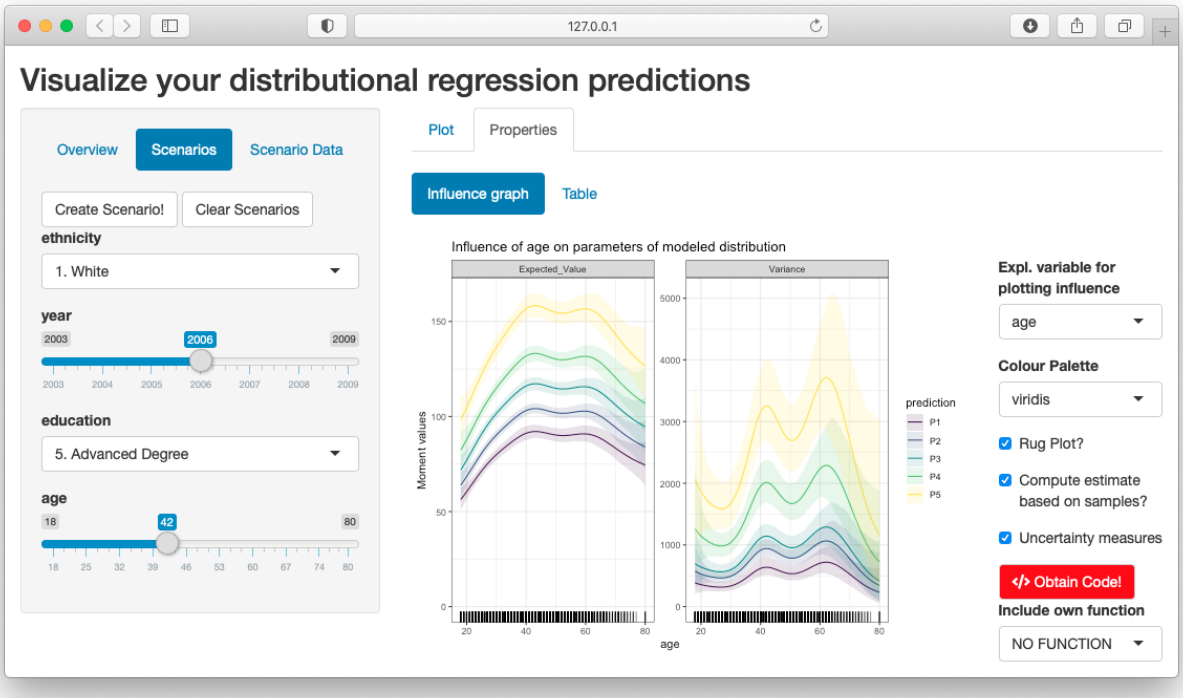

Figure B8: Influence of `age` on the first two moments of the predicted distributions for `wage_model`.

next to a small bar with additional options. On the right side, the continuous variable `age` was selected. This leads to a recreation of Figure A2 part a) without the specification of the external function. Selecting the categorical variable `ethnicity` would lead to part b) of Figure A2.

Next to the plot in the "Influence graph" tab multiple web elements are displayed. The first one, located under the description "Expl. variable for plotting influence", lets the user select the explanatory variable for which the influence plot shall be created. The second element gives options to choose a color palette, similar to Figure B6. In Figures B8 the "viridis" color palette was specified to account for potential colorblindness. Below the palette selector the same red button as in Figure B6 is placed which when pressed presents code to reproduce the influence plot with (Section D: Figure D14).

At the bottom end of the web elements, we see a selector element titled "Include own function", which corresponds to the `plot_moments()` argument `ex_fun`. If a user-written function calculating a measure based on the expected parameters of the target distribution is specified here, the influence of the selected covariate on this measure is calculated and

included into the plot (similar to part a) of Figure A2).

The other sub-tab of "Properties" called "Table" is useful if the user solely wants to see the differences in distributional moment values across specified covariate combinations in the "Scenarios tab", without it depending on a selected covariate's range. Figure D15 shows this tab's layout.

The only present element in this tab is a table showing two columns, `Expected_Value` and `Variance` with computed values for the first two moments. Every row depicts one covariate combination specified in the "Scenario" tab.

C Special Case of plot_dist()

One more special case exists within using `plot_moments()`, which appears when the family of multinomial distributions is used for describing the dependent variable. Due to the nature of the variable's parameters π_i as the probability to fall into category i always summing up to 1, we can visualize the impact of a continuous variable differently. Figure C9 shows such a case, where the influence of a continuous variable named `norm1` on a categorical variable, both stemming from a simulated dataset, was graphically displayed.

The three windows in Figure C9 represent three different covariate combinations specified in `pred_data`. On the x-axis, we see the range of `int_var`, while the y axis denotes probabilities. Every colored area represents the expected probability for a new observation to fall into any of the categories of the dependent variable. Due to the target distribution being highly dependent on the simulated explanatory variables, its probabilities change considerably over the range of the variable `norm2`.

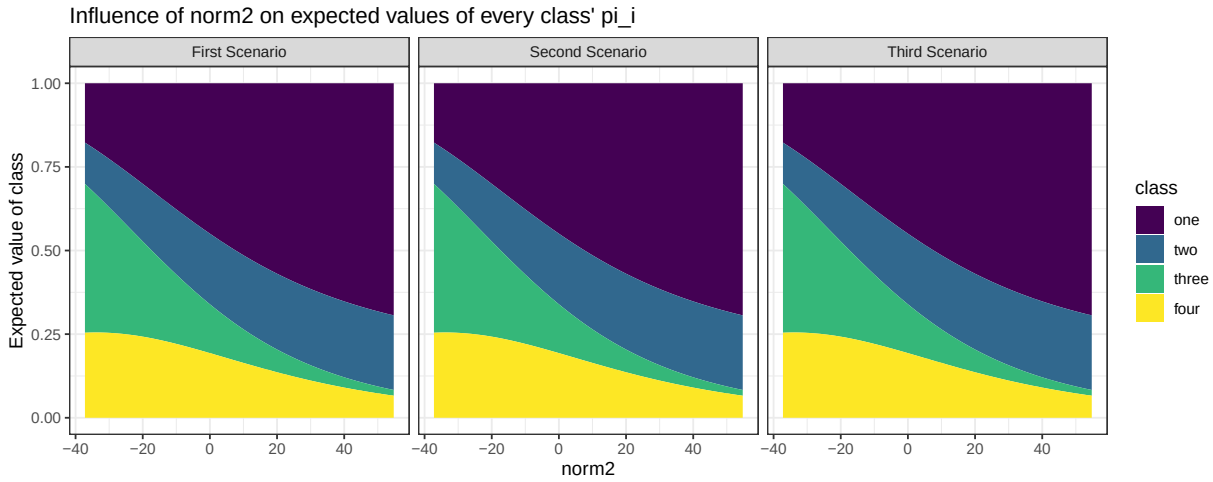

Figure C9: Outcome of `plot_moments()` for a model with a simulated multinomial target distribution. In this case, each scenario gets its own window, with the expected probabilities to fall into each category ranging over the variable of interest.

D Additional Graphs

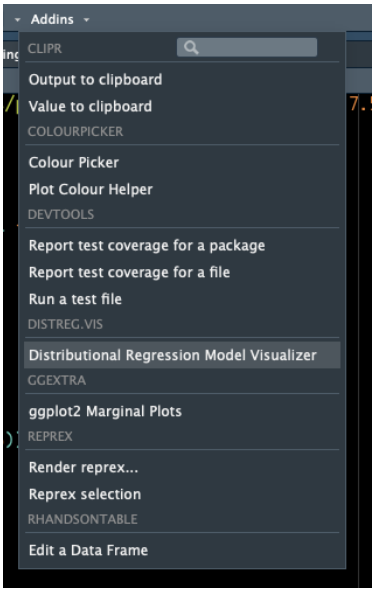

Figure D10: Button to start the main application of distreg.vis in RStudio.

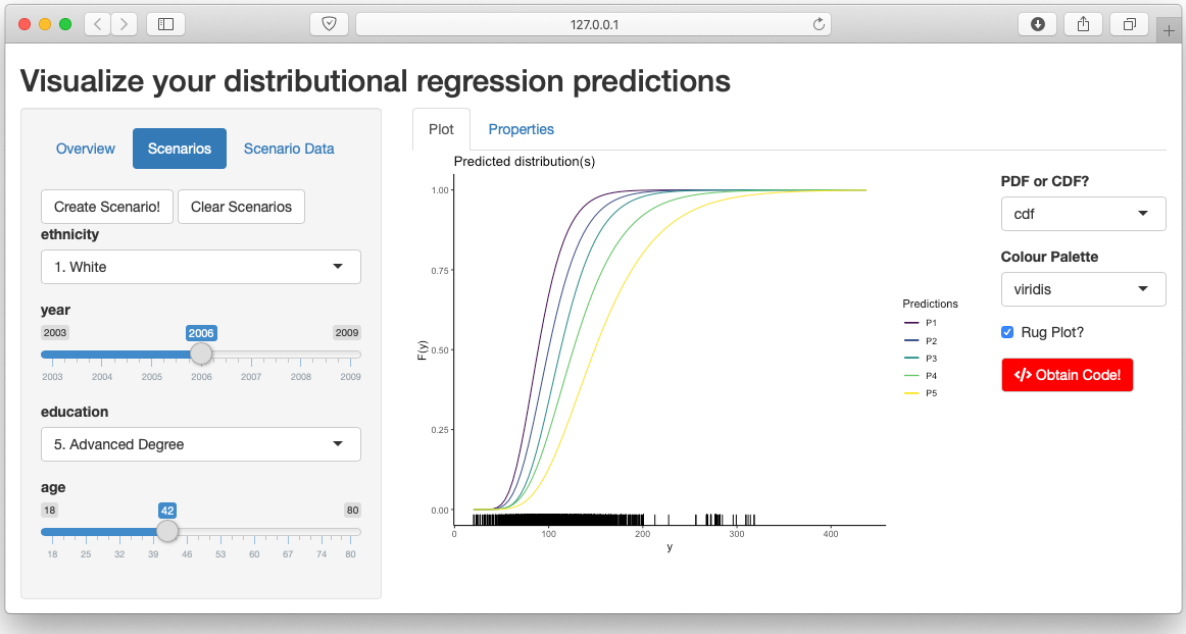

Figure D11: Cumulative Distribution Function plot output for different education levels based on the `Wage` dataset.

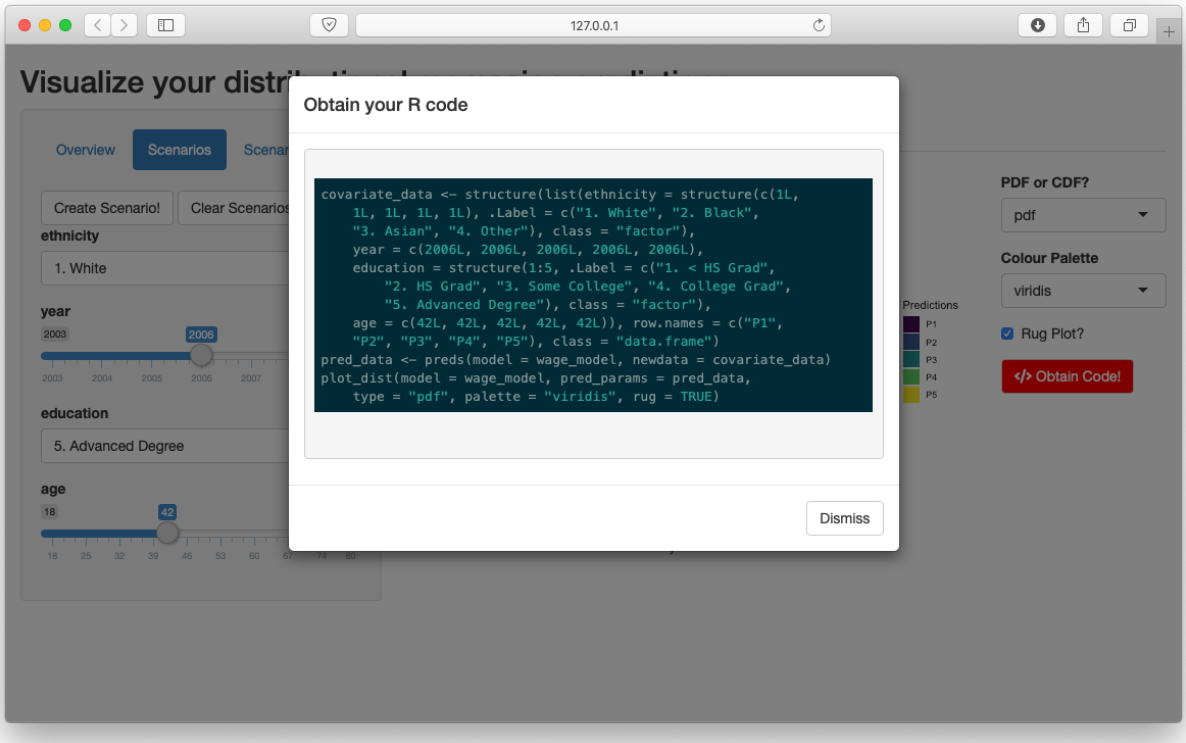

Figure D12: Modal window with formatted and highlighted code after pressing the "Obtain Code!" button

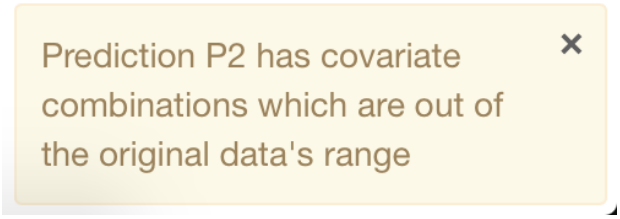

Figure D13: Warning message when specifying covariate combinations which are out of range.

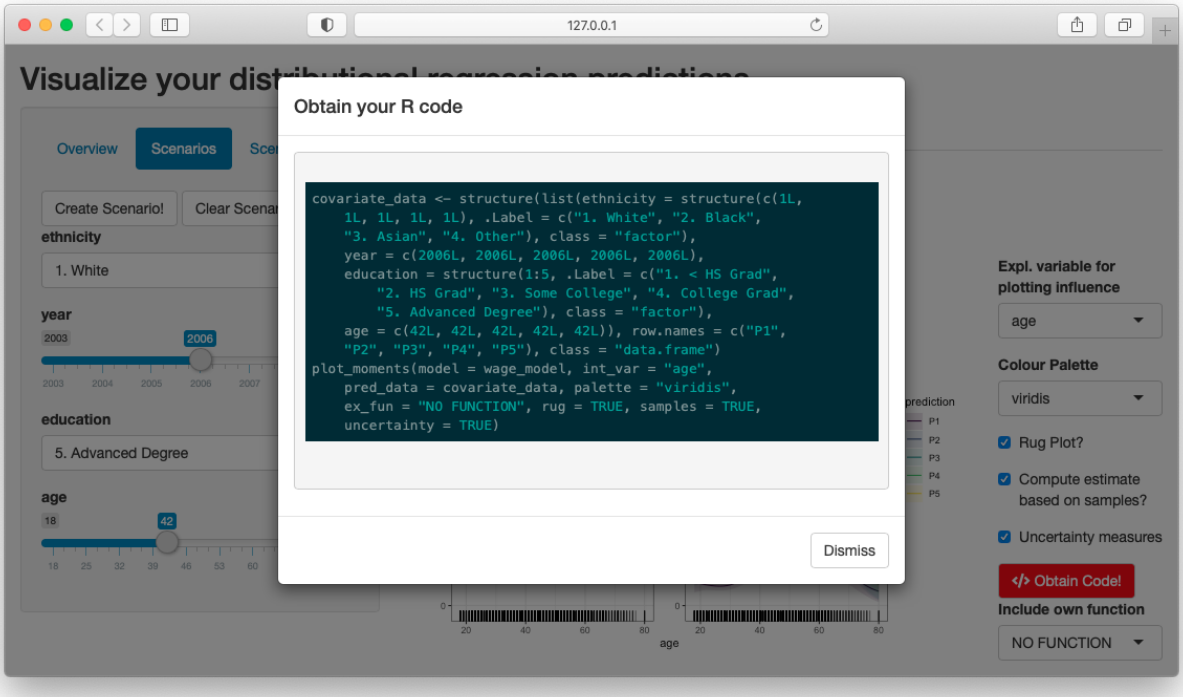

Figure D14: Modal window to display code for reproducing the influence plot.

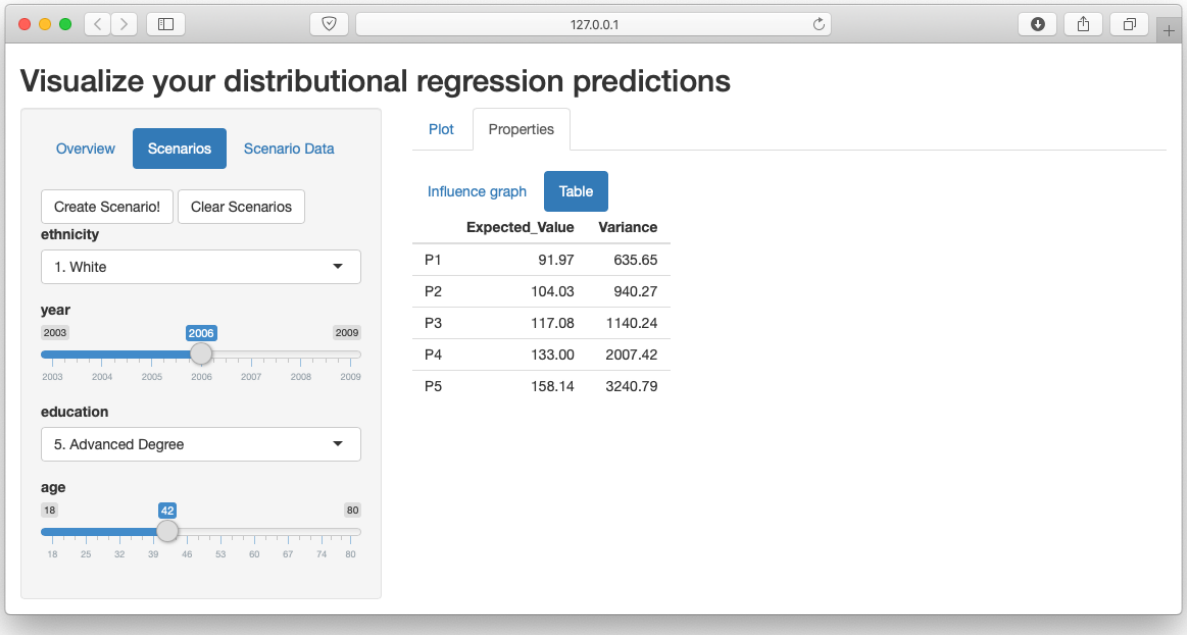

Figure D15: Expected Value and Variance for predicted distributions based on specified covariate combinations.

References

S. Garnier. *viridis: Default Color Maps from 'matplotlib'*, 2017. URL https://CRAN.R-project.org/package=viridis. R package version 0.4.0.

R. I. Lerman and S. Yitzhaki. A note on the calculation and interpretation of the gini index. *Economics Letters*, 15(3-4):363–368, 1984.

E. Neuwirth. *RColorBrewer: ColorBrewer Palettes*, 2014. URL https://CRAN.R-project.org/package=RColorBrewer. R package version 1.1-2.

J. Owen. *rhandsontable: Interface to the 'Handsontable.js' Library*, 2016. URL https://CRAN.R-project.org/package=rhandsontable. R package version 0.3.4.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL http://ggplot2.org.