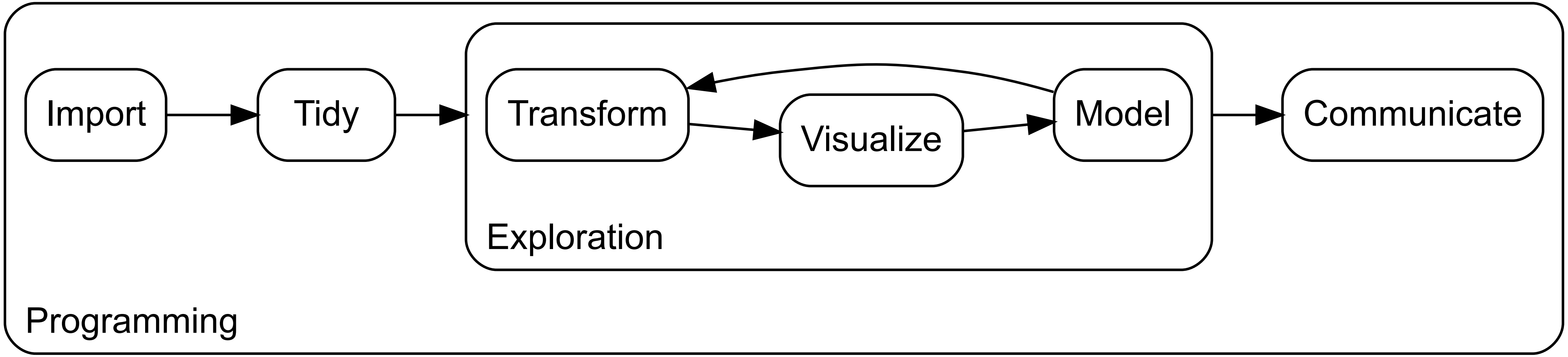
Data Analysis using R

Modeling

Sven Werenbeck-Ueding

25.09.2023



Source: Wickham and Grolemund (2016)

Fenced Out:

The Impact of Border Construction on U.S.-Mexico Migration

> This paper estimates the **impact of the US-Mexico border fence on US-Mexico migration** by **exploiting variation in the timing and location of US government investment in fence construction**. Using Mexican survey data and data I collected on fence construction, I find that construction in a municipality reduces migration by 27 percent for municipality residents and 15 percent for residents of adjacent municipalities.
>
> Feigenberg (2020a)

Data

Mexican Survey

- Encuesta Nacional de Ocupación y Empleo (ENOE) from Q3 2003 to Q3 2013
- Quarterly rotating panel: Households included for 5 quarters
- Records whether any household member leaves to the US
- Potential migrants are restricted from ages 15 to 65
- Explanatory variables: age, gender, marital status and education for all household members

Fence Construction

- Data collected by identifying potential fence locations:
  - Documents from US authorities: Customs and Border Protection (CPB) and Government Accountability Office (GAO)
  - Local government reports
  - Published contracts with construction firms
- Cross-checked with an environmental organization tracking the impact of fence construction (Sierra Club)
- Provides quarterly information on fence construction on Mexican municipality level

Identification Strategy

- By exploiting temporal and spatial variation in fence construction, Feigenberg (2020a) estimates the impact of fence construction on the migration decision of potential migrants
- Exogenous variation in fence construction allows for a difference-in-difference estimator of the form

$$Pr(Y_{mti} = 1 \mid z_{mti}) = \frac{exp(\alpha + X_{mti}\beta + \delta \times fence_{mti} + \gamma_m + \tau_t)}{1 + exp(\alpha + X_{mti}\beta + \delta \times fence_{mti} + \gamma_m + \tau_t)}$$

  - $Y_{mti} \in [0, 1]$: 1 if individual i living in municipality m migrates to the US in year-quarter t
  - X_{mti}: Socio-economic characteristics of individual i living in municipality m and year-quarter t
  - $fence_{mti} \in [0, 1]$: 1 if fence construction started in municipality m in or before year-quarter t
  - γ_m, τ_t: Municipality and year-quarter fixed effects

Results

- Table 2 shows log-odds coefficients of different specifications for the estimation strategy
- Column (2) of panel A employs the specification on the previous slide
- Fence construction reduces probability of migrating by $1 - e^{\delta} = 37.87\%$
- Relative to baseline migration rate of 4.2 per $1,000$ respondents
- Parentheses show standard errors clustered by municipality
- Effect is highly significant: p-value is ≈ 0

Table 2—Impact of Fence Construction on Border Municipality Migration

	Migrate to United States		
	(1)	(2)	(3)
Panel A			
Fence construction	–0.319	–0.476	–0.447
	(0.129)	(0.132)	(0.192)
Panel B			
Fence construction	–0.283	–0.398	–0.548
	(0.136)	(0.158)	(0.168)
Number of adjacent municipalities fenced	–0.164	–0.181	–0.318
	(0.0891)	(0.0909)	(0.110)
Number of fenced municipalities two away	0.0389	–0.0490	–0.0216
	(0.0708)	(0.0933)	(0.135)
Panel C			
Fence construction	–0.319	–0.438	–0.665
	(0.178)	(0.212)	(0.242)
Number of adjacent municipalities fenced	–0.192	–0.211	–0.401
	(0.120)	(0.129)	(0.193)
Number of fenced municipalities two away	0.0440	–0.0443	0.0164
	(0.0752)	(0.0961)	(0.153)
Fence construction × number of adjacent municipalities fenced	0.0520	0.0566	0.167
	(0.118)	(0.132)	(0.171)
Observations	330,503	316,591	316,591
Municipality fixed effects	X	X	X
Year-quarter fixed effects	X	X	X
Additional controls		X	X
State × year-quarter fixed effects			X
Mean of non-fenced		0.00420	
		[0.0647]	

Source: Feigenberg (2020a)

Prerequisites

```
# install.packages("tidymodels")
library(tidyverse)
library(tidymodels)

df_mig <- read_csv("data/processed/fence_migration.csv")
```

- For this course, a random 50% sample of the full sample was drawn from the data set provided by Feigenberg (2020b)
- Our results may differ

Regressions in R

Fit Linear Regressions in R

```
?lm
```

The `lm()` function fits linear models, including multivariate models. It can also be used to carry out single stratum analysis of variance and analysis of covariance.

- `formula`: An object of class `formula` that symbollically describes the model to be fitted
- `data`: An object of class `data.frame` (or coercible by `as.data.frame`) containing the model variables
- `subset`: An optional vector for subsetting observations in `data`
- `weights`: An optional numeric vector of weights, e. g. for weighted least squares

Formulas in R

Expressions such as `y ~ x1 + x2 + x3` use the ~ operator to specify that response `y` is modeled by a set of predictors (`x1`, `x2` and `x3`)

Operator	Meaning	Example
`:`	Interaction effect between two predictors	`x1:x2`
`*`	Main and interaction effects of predictors	`x1*x2` → `x1 + x2 + x1:x2`
`^`	Expands to a formula containing main effects and interactions up to the n^{th} order	`(x1 + x2 + x3)^2` → `x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3`
`/`	Terms on the LHS are nested within those on the right	`x1/x2` → `x1 + x1:x2`
`-`	Removes terms from the formula (e. g. the intercept)	`y ~ -1 + x1`
`.`	Usually interpreted as all `data` columns not otherwise in the formula	`y ~ .`

Binary Choice Models

Model	Functional Form	Command
LPM	$Pr(Y_i = 1 \mid X_i) = X_i\beta$	`lm(y ~ .)`
Logit	$Pr(Y_i = 1 \mid X_i) = \wedge(X_i\beta) = \dfrac{e^{X_i\beta}}{1 + e^{X_i\beta}}$	`glm(y ~ ., family = "binomial")`
Probit	$Pr(Y_i = 1 \mid X_i) = \phi(X_i\beta) = \int_{-\infty}^{X_i\beta} \phi(z)dz$	`glm(y ~ ., binomial(link = "probit"))`

Task 1: Replicate the Estimation of Feigenberg (2020a)

Task Code Output

Estimate the impact of border fence construction on Mexican-US migration. Implement the logit model employed by Feigenberg (2020a) in R and interpret the results. Use the data provided in `data/processed/fence_migration.csv`.

$$Pr(Y_{mti} = 1 \mid z_{mti}) = \wedge\,(\alpha + X_{mti}\beta + \delta \times fence_{mti} + \gamma_m + \tau_t)$$

- $Y_{mti} \in [0, 1]$: 1 if individual i living in municipality m migrates to the US in year-quarter t
- X_{mti}: Socio-economic characteristics of individual i living in municipality m and year-quarter t
- $fence_{mti} \in [0, 1]$: 1 if fence construction started in municipality m in or before year-quarter t
- γ_m, τ_t: Municipality and year-quarter fixed effects

Task 1: Replicate the Estimation of Feigenberg (2020a)

Task | Code | Output

```r
# Define the regression formula
full_formula <- formula(
  migrate ~ fence + female + age + educ + married +
    as.factor(municipality) + as.factor(period)
)

# Fit a logit model as given by formula on the df_mig data set
logit_model <- glm(full_formula, df_mig, family = "binomial")

# Print a tidy model summary to the console
broom::tidy(logit_model) %>%
  filter(term %in% c("fence", "female", "age", "educ", "married"))
```

Task 1: Replicate the Estimation of Feigenberg (2020a)

Task Code **Output**

```
## # A tibble: 5 × 5
##   term     estimate std.error statistic  p.value
##   <chr>       <dbl>     <dbl>     <dbl>    <dbl>
## 1 fence     -0.471    0.207       -2.28 2.28e- 2
## 2 female    -0.473    0.0910      -5.19 2.07e- 7
## 3 age       -0.0166   0.00390     -4.25 2.13e- 5
## 4 educ       0.0119   0.0115       1.04 2.98e- 1
## 5 married   -0.626    0.101       -6.18 6.28e-10
```

Econometricians seldomly estimate just one model!

Different model specifications serve different purposes

- Including additional controls for robustness checks
- Investigating heterogeneity by interacting variables
- Transforming variables, e. g. `log(y)`

In some cases, specifications are estimated using different models

- Estimating logit and LPM for robustness check
- Finding the best predictor among a set of models (common in machine learning applications)

Writing clean and easily understandable code becomes difficult

- R implementations of models have different interfaces
- Preprocessing steps are not easily interchangable using base R

tidymodels

> The `tidymodels` framework is a collection of packages for modeling and machine learning using `tidyverse` principles.
>
> Kuhn and Wickham (2020)
>
> These two aspects of model development – **ease of proper use** and **good methodological practice** – are crucial. [...] Tools should be powerful enough to create high-performance models, but, on the other hand, should be easy to use appropriately.
>
> Kuhn and Silge (2022)

Taxonomy of Model Types

Descriptive

- Illustrates characteristics of the data
- E. g. visualize data to identify relationships between variables
- Discover ways to represent variables in a model
- Almost always precedes other types of models

Inferential

- Explores hypotheses on the relation between variables
- Produces probabilistic output to find some statistical conclusion, e. g. t-tests
- Acceptance or rejection of hypothesis depends on pre-defined assumptions

Predictive

- Estimating new data on a pre-trained model to predict values as close as possible to the new values
- Less concerned with how variables are related in the model and more *empirically* driven

parsnip

> The goal of parsnip is to provide a tidy, unified interface to models that can be used to try a range of models without getting bogged down in the syntactical minutiae of the underlying packages.
>
> Kuhn and Vaughan (2022)

Why use `parsnip`?

1. Separate the model definition from its evaluation

- Same model is often re-run with different preprocessing steps
- E. g. baseline specification and specification with additional controls

2. Detach model specification from implementation

- Different engines could be used to fit a linear model, e. g. `lm`, `glm` etc.
- Makes comparison between engines easier

3. Provide consistent naming of function arguments

→ `parsnip` offers a unified approach to model specifications

Components of Model Specifications

1. Model Type

Specify the model type to use (e. g. `linear_reg()` for linear regression, `logistic_reg` for logit, …)

2. Engine

Set the engine for fitting the model. Run `show_engines("linear_reg")` to get a list of engines (and their modes)

3. Mode

When required, determine the model's mode. Numeric outcomes are estimated by use of regression, whereas qualitative outcomes require a classification model. For some models, e. g. linear regression, only one mode is available and therefore automatically set (regression).

Task 2: Logit Model Specification

Task Code Output

Revisit the replication of Feigenberg (2020a)'s identification strategy in Task 1. This time, use the `parsnip` package for specifying and fitting the model. Print a summary of your model to the console.

Task 2: Logit Model Specification

Task | **Code** | Output

```r
# Specify type and engine of the logit model
logit_model <- logistic_reg() %>%
  set_engine("glm")

# Fit the model in a separate step using the formula
formula_full <- formula(
  as.factor(migrate) ~ fence + female + age + educ + married +
    as.factor(municipality) + as.factor(period)
)

logit_fit_full <- logit_model %>%
  fit(formula_full, data = df_mig)

# Retrieve the fitted glm object and print a summary
tidy(logit_fit_full) %>%
  filter(term %in% c("fence", "female", "age", "educ", "married"))
```

Task 2: Logit Model Specification

Task Code **Output**

```
## # A tibble: 5 × 5
##   term    estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 fence    -0.471    0.207       -2.28 2.28e- 2
## 2 female   -0.473    0.0910      -5.19 2.07e- 7
## 3 age      -0.0166   0.00390     -4.25 2.13e- 5
## 4 educ      0.0119   0.0115       1.04 2.98e- 1
## 5 married  -0.626    0.101       -6.18 6.28e-10
```

Task 3: Fit Different Specification

Task | Code | Output

Use the same model type and engine as in Task 2, but exclude the additional controls in your regression formula:

$$Pr(Y_{mti} = 1 \mid z_{mti}) = \wedge \left(\alpha + \delta \times fence_{mti} + \gamma_m + \tau_t\right)$$

Task 3: Fit Different Specification

Task | **Code** | Output

```
# Specify the formula without additional controls
formula_reduced <- formula(
  as.factor(migrate) ~ fence + as.factor(municipality) + as.factor(period)
)

# Fit the formula to the logit model defined before
logit_reduced_fit <- logit_model %>%
  fit(formula_reduced, data = df_mig)

# Print a summary of the fitted glm model
tidy(logit_reduced_fit) %>%
  filter(term == "fence")
```

Task 3: Fit Different Specification

Task Code **Output**

```
## # A tibble: 1 × 5
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 fence   -0.449     0.206     -2.18  0.0294
```

Estimating Multiple Specifications

- As seen in Task 3, `parsnip` allows for greater flexibility in modeling than `base` R
  - Various specifications can be run on the same model
  - `base` R models would have to be specified independently
  - Model engine can be easily exchanged as well
- Especially useful when estimating a multitude of different specifications
- Modeling via `parsnip` can be natively used in functional programming (see the code on the right)

```
logit_model <- logistic_reg() %>%
  set_engine("glm")

formulas <- list(
  reduced = formula(
    as.factor(migrate) ~ fence +
      as.factor(municipality) +
      as.factor(period)
  ),
  full = formula(
    as.factor(migrate) ~ fence + female +
      age + educ + married +
      as.factor(municipality) +
      as.factor(period)
  )
)

logit_fit <- formulas %>%
  map(~ fit(logit_model, ., data = df_mig))
```

`workflows`

> Managing both a `parsnip` model and a preprocessor, such as a model formula or recipe from `recipes`, can often be challenging. The goal of `workflows` is to streamline this process by bundling the model alongside the preprocessor, all within the same object.
>
> Vaughan and Couch (2022)

What constitutes a model?

> **Model fitting** is often **considered to be the only step in the modeling** process. Non-trivial data structures and models with high(er) complexity, however, require additional steps:

Pre-Processing

- Selecting predictors from a set of candidates
  - Exploratory data analysis
  - Domain knowledge
  - Data-driven algorithms
- Imputation of missing values
- Transforming the scale of a predictor

Post-Processing

- Transforming estimates into interpretable format, e. g. log-odds for logit regressions
- Adjusting standard errors

Workflow Basics

```
?workflow
```

> A `workflow` is a container object that aggregates information required to fit and predict from a model. Workflows *always* require a `parsnip` model object **and** a preprocessor.

Arguments:

Argument	Description
`preprocessor`	A preprocessor used for processing the data prior to fitting the model. Can be an object of class `formula`, specifying the variables in a model.
`spec`	A `parsnip` model specification.

Creating a Workflow

Code Output

- Use the formula incl. additional controls defined on previous slides as `preprocessor`
- Set the workflow's model to the `parsnip` model object

```
# Add preprocessor and model to a workflow
logit_full_wflow <- workflow(
  preprocessor = formula_full,
  spec = logit_model
)

# Print the workflow object
logit_full_wflow
```

Creating a Workflow

Code Output

```
## ══ Workflow ════════════════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: logistic_reg()
##
## ── Preprocessor ────────────────────────────────────────────────────────────────
## as.factor(migrate) ~ fence + female + age + educ + married +
##     as.factor(municipality) + as.factor(period)
##
## ── Model ───────────────────────────────────────────────────────────────────────
## Logistic Regression Model Specification (classification)
##
## Computational engine: glm
```

Fitting a Workflow

Code | Output

- `fit()` the workflow on the `df_mig` data set
- Returns a workflow object with a fitted parsnip model in the `.$fit$fit` of the workflow object

```
# Fit the workflow to the CPR data
logit_full_fit <- fit(logit_full_wflow,
                      df_mig)

# Use the tidy command to get a tibble of
# estimated coefficients
logit_full_fit %>%
  tidy() %>%
  filter(
    term %in% c(
      "fence", "female", "age", "educ",
      "married"
    )
  )
```

Fitting a Workflow

Code Output

```
## # A tibble: 5 × 5
##   term    estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 fence    -0.471    0.207       -2.28 2.28e- 2
## 2 female   -0.473    0.0910      -5.19 2.07e- 7
## 3 age      -0.0166   0.00390     -4.25 2.13e- 5
## 4 educ      0.0119   0.0115       1.04 2.98e- 1
## 5 married  -0.626    0.101       -6.18 6.28e-10
```

`recipes`

> With recipes, you can use `dplyr`-like pipeable sequences of feature engineering steps to get your data ready for modeling.
>
> Kuhn and Wickham (2022)

Recipe Basics

```
?recipes::recipe
```

> A recipe is a description of the steps to be applied to a data set in order to prepare it for data analysis.

It defines:

1. Variables

Data columns in a `data.frame` or `tbl`.

2. Roles

Definition of how variable are used in a model, most commonly `outcome` or `response`.

3. Terms

Columns in a design matrix, such as `educ` or `educ:female`. Variables with the role `predictor` are automatically main effect terms.

Recipe Basics

```
?recipes::recipe
```

> A recipe is a description of the steps to be applied to a data set in order to prepare it for data analysis.

Arguments:

Argument	Description
`formula`	A model formula **without** in-line functions, e. g. `log()`. In-line functions that transform the data can be passed to the recipe using `step_*()` functions.
`data`	A data frame or tibble of the *template* data set. `data` does not have to be the actual data, but must have the same names and types of the target data set. For large data sets, it is sufficient to simply pass `head(data)` to the recipe.

Creating a Recipe

Code | Output

- Before creating the preprocessor, coerce `migrate`, `municipality` and `period` to factor (needed for estimation)
- Can not be done as in-line function in the formula

```r
# Coerce columns to factor
fct_cols <- c("migrate", "municipality",
              "period")

df_mig <- df_mig %>%
  mutate(across(all_of(fct_cols),
                ~ forcats::as_factor(.)))

formula_full <- migrate ~ fence + female +
  age + educ + married +
  municipality + period

# Create a recipe by providing a formula and
# data frame for variable selection
rec_full <- recipe(formula_full, df_mig)

rec_full
```

Creating a Recipe

Code Output

```
## Recipe
##
## Inputs:
##
##        role #variables
##     outcome          1
##   predictor          7
```

Adding Preprocessing Steps to a Recipe

- Recipes can be piped into step functions called `step_*()`
- Step functions are preprocessing operations on the data that can be added sequentially to a recipe *without* being immediately executed
- `dplyr`-like selector functions can be used to select columns to operate on
- Offers greater flexibility than specifying e. g. variable transformations directly in the formula

Adding Preprocessing Steps to a Recipe

General function structure:

```
step_*(
  recipe,
  ...
)
```

Argument	Description
`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One (or more) selector functions to choose variables (see `?selections` for more details).

Adding Preprocessing Steps to a Recipe

Step Function	Description
`step_dummy()`	Converts nominal data into numeric binary model terms.
`step_log()`	Log transforms variables.
`step_interact()`	Creates new columns for interaction terms between variables. Terms for which interactions should be created are passed to the `terms` argument using a formula containing interactions or selectors.
`step_mutate()`	Adds variables using `dplyr::mutate()`.
`step_filter()`	Removes rows using `dplyr::filter()`.
`step_select()`	Selects variables using `dplyr::select()`.
`step_naomit()`	Removes rows containing `NA` or `NaN`.

Note: See the `recipes` reference for a complete overview of step functions included in the package.

Selector Functions for Preprocessing Steps

```
?recipes::selections
```

> When selecting variables or model terms in step functions, `dplyr`-like tools are used. The selector functions can choose variables based on their name, current role, data type, or any combination of these.

Selection	Description
`all_numeric()`	Selection based on type
`all_nominal()`	Selection based on type
`all_predictors()`	Selection based on role
`all_outcomes()`	Selection based on role
`all_numeric_predictors()`	Selection based on type and role
`all_nominal_predictors()`	Selection based on type and role

Note: `recipes` also supports select helpers from the `tidyselect` package, such as `everything()`, `all_of()` and `any_of()`.

Preprocessing the Migration Data

Code | Output

- Assign roles to the variables
  - `migrate` is the outcome
  - As before, `fence`, some of the socio-economic characteristics, `municipality` and `period` are the explanatory variables ("predictors")
- When fitting the recipe, the variables are included in the estimation as defined by their roles

```
predictors <- c("fence", "female", "age",
                "educ", "married",
                "municipality", "period")

# Create a preprocessor by specifying the
# data set and assign roles
rec_full <- recipe(df_mig) %>%
  update_role(migrate,
              new_role = "outcome") %>%
  update_role(all_of(predictors),
              new_role = "predictor")

# Print preprocessor
rec_full
```

Preprocessing the Migration Data

Code Output

```
## Recipe
##
## Inputs:
##
##         role #variables
##      outcome          1
##    predictor          7
##
##   4 variables with undeclared roles
```

Fitting the Model with a Recipe

Code Output

```
# Create a workflow with the preprocessor and the model specification
logit_full_wflow <- workflow(preprocessor = rec_full,
                             spec = logit_model)

# Fit the workflow
logit_full_fit <- fit(logit_full_wflow, data = df_mig)

# Print a model summary
tidy(logit_full_fit) %>%
  filter(term %in% c("fence", "female", "age", "educ", "married"))
```

Fitting the Model with a Recipe

Code Output

```
## # A tibble: 5 × 5
##   term    estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 age      -0.0166   0.00390     -4.25 2.13e- 5
## 2 educ      0.0119   0.0115       1.04 2.98e- 1
## 3 female   -0.473    0.0910      -5.19 2.07e- 7
## 4 married  -0.626    0.101       -6.18 6.28e-10
## 5 fence    -0.471    0.207       -2.28 2.28e- 2
```

Task 4: Fit Multiple Models

Task | Code | Basic Specification | Full Specification

Re-run the estimations from Task 2 and Task 3 but use workflows to fit your models. Additionally, make use of the functions provided in the `purrr` package.

Task 4: Fit Multiple Models

Task | **Code** | Basic Specification | Full Specification

```r
# Store recipes in a list
recs <- list(reduced = rec_full %>%
               step_select(-age, - female, -educ, -married),
             full = rec_full)

# Specify the model
logit_model <- logistic_reg() %>%
  set_engine("glm")

# Map the preprocessors on workflows and fit the models
models_fit <- recs %>%
  map(~ workflow(., spec = logit_model)) %>% # Create a workflow for each list element
  map(~ fit(., data = df_mig))                # Fit each workflow stored in the list
```

Task 4: Fit Multiple Models

Task | Code | **Basic Specification** | Full Specification

```
## # A tibble: 1 × 5
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 fence   -0.449     0.206     -2.18  0.0294
```

Task 4: Fit Multiple Models

Task | Code | Basic Specification | Full Specification

```
## # A tibble: 5 × 5
##   term    estimate std.error statistic  p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 age      -0.0166   0.00390     -4.25 2.13e- 5
## 2 educ      0.0119   0.0115       1.04 2.98e- 1
## 3 female   -0.473    0.0910      -5.19 2.07e- 7
## 4 married  -0.626    0.101       -6.18 6.28e-10
## 5 fence    -0.471    0.207       -2.28 2.28e- 2
```

Extract the Engine Fit

> The generic function `extract_fit_engine()` returns the underlying fitted model object. For a `parsnip` model with the `"lm"` engine, i. e. a `lm` object:

```
logit_full_fit <- extract_fit_engine(models_fit$full)

class(logit_full_fit)
```

```
## [1] "glm" "lm"
```

→ Necessary when next steps in the data analysis process, e. g. creating regression tables, require objects of a certain class as an input factor

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`models`

A model object, such as `lm`, or a (named) list of models.

Regression Tables

A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`output`

If the table should be saved directly, the filename can be specified here. If the table should be customized afterwards, this argument should be set to the desired object type, e. g. `"kableExtra"`.

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```r
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`vcov`

Replace model standard errors with robust standard errors by setting this argument to `"HC3"` or other variants of heteroscedasticity-consistent standard errors. Including robust standard errors in the table requires the `sandwich` package.

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`stars`

Show stars to indicate statistical significance by passing a named numeric vector to this argument. Most commonly, this would be set to:

```
c("*" = .1, "**" = .05, "***" = .01)
```

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`title`

Title for the table given as a string.

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`notes`

Pass a list or vector of strings to this arguments to show notes below the table.

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`coef_map`

Use a named character vector to map model variable names to coefficient names, e. g. `c(female = "Female")`. Coefficients are shown in the order of the character vector. If a coefficient is not given in the vector, it will be omitted from the table.

Regression Tables

> A variety of packages offers solutions to create off-the-shelve tables for regression output. The `msummary()` from the `modelsummary` package takes an `lm` object – or a (named) list of `lm` objects – as input and returns a customizable regression table.

```
msummary(
  models,
  output = "default",
  vcov = NULL,
  stars = FALSE,
  title = NULL,
  notes = NULL,
  coef_map = NULL,
  gof_map = NULL,
  ...
)
```

`gof_map`

A character vector specifying goodness-of-fit statistics and other model information to show at the bottom of the table. Measures are reported in the order given in the vector. See `get_gof(<YOUR-MODEL>)` for a list of measures to choose from. Names of the `data.frame` correspond to measure names that have to be provided to show in the table.

If you want to use a custom name for the measures, you can pass a `data.frame` object with the columns `"raw"`, `"clean"` and `"fmt"` instead of a character vector.

Task 5: Create a Summary Table Showing Your Results

Task | Code

Use the `msummary()` function from the `modelsummary` package to create a table showing the results from the models estimated in Task 4.

Show only coefficients for your variable of interest and the additional controls and cluster standard errors on municipality level using the `vcovCL()` function from the `sandwich` package.

Task 5: Create a Summary Table Showing Your Results

Task | **Code**

```r
library(modelsummary)

models_fit %>%
  map(extract_fit_engine) %>%
  set_names(c("(1)", "(2)")) %>%
  msummary(
    output = "kableExtra", # May also be e. g. "latex" or a filename extension such as ".txt"
    # Use sandwich package to calculate clustered vcov matrix
    vcov = function(x) sandwich::vcovCL(x, cluster = df_mig$municipality),
    stars = c("*" = .1, "**" = .05, "***" = .01),
    title = "Impact of Fence Construction on Mexian-US Migration",
    notes = "Parantheses show clustered standard errors.",
    coef_map = c(fence = "Fence Construction", age = "Age", female = "Female",
                 educ = "Years of Schooling", married = "Married"),
    gof_map = c("nobs")
  )
```

Impact of Fence Construction on Mexian-US Migration

	(1)	(2)
Fence Construction	-0.449**	-0.471**
	(0.212)	(0.211)
Age		-0.017***
		(0.005)
Female		-0.473***
		(0.098)
Years of Schooling		0.012
		(0.021)
Married		-0.626***
		(0.104)
Num.Obs.	156113	156113
* p < 0.1, ** p < 0.05, *** p < 0.01		
Parantheses show clustered standard errors.		

References

Feigenberg, B. (2020a). "Fenced Out: The Impact of Border Construction on US-Mexico Migration". In: *American Economic Journal: Applied Economics* 12.3, pp. 106-39. DOI: 10.1257/app.20170231. URL: https://www.aeaweb.org/articles?id=10.1257/app.20170231.

Feigenberg, B. (2020b). *Replication package for: Fenced Out: The Impact of Border Construction on US-Mexico Migration.* American Economic Association. URL: https://www.aeaweb.org/journals/dataset?id=10.1257/app.20170231.

Kuhn, M. and J. Silge (2022). *Tidy Modeling with R. A Framework for Modeling in the Tidyverse*. O'Reilly. URL: https://www.tmwr.org/.

Kuhn, M. and D. Vaughan (2022). *parsnip: A Common API to Modeling and Analysis Functions*. URL: https://parsnip.tidymodels.org/.

Kuhn, M. and H. Wickham (2020). *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles.* URL: https://www.tidymodels.org.

Kuhn, M. and H. Wickham (2022). *recipes: Preprocessing and Feature Engineering Steps for Modeling*. URL: https://recipes.tidymodels.org/.

References

Vaughan, D. and S. Couch (2022). *workflows: Modeling Workflows*. URL: https://workflows.tidymodels.org/.

Wickham, H. and G. Grolemund (2016). *R for data science. import, tidy, transform, visualize, and model data*. O'Reilly. URL: https://r4ds.had.co.nz/.