In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab6-regression.ipynb")
```

In [ ]:
```
import numpy as np
import pandas as pd
import altair as alt
import statsmodels.api as sm

# disable row limit for plotting
alt.data_transformers.disable_max_rows()
# uncomment to ensure graphics display with pdf export
# alt.renderers.enable('mimetype')
```

Lab 6: Regression

This lab covers the nuts and bolts of fitting linear models. The linear model expresses a response variable, y, as a linear function of $p-1$ explanatory variables $x_1, \ldots, x_{p-1}$ and a random error ϵ. Its general form is:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_{p-1} x_{p-1} + \epsilon \qquad \epsilon \sim N(0, \sigma^2)$$

Usually, the response and explanatory variables and error term are indexed by observation $i = 1, \ldots, n$ so that the model describes a dataset comprising n values of each variable:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_{p-1} x_{i,p-1} + \epsilon_i \qquad \begin{cases} \epsilon_i \sim N(0, \sigma^2) \\ i = 1, \ldots, n \end{cases}$$

Because the indices get confusing to keep track of, it is much easier to express the model in matrix form as

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

where:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1} \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1,p-1} \\ 1 & x_{21} & \cdots & x_{2,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{n,p-1} \end{bmatrix}_{n \times p} \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix}_{p \times 1} \qquad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Fitting a model of this form means ***estimating the parameters*** $\beta_0, \beta_1, \ldots, \beta_{p-1}$ and σ^2 from a set of data.

- The ordinary least squares (OLS) estimates of β, which are best under most circumstances, are

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

- The error variance σ^2 can be estimated by

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\left(\mathbf{y} - \mathbf{X}\hat{\beta}\right)'\left(\mathbf{y} - \mathbf{X}\hat{\beta}\right)$$

When fitting a linear model, it is also of interest to quantify uncertainty by estimating the variability of $\hat{\beta}$ and measure overall quality of fit. This lab illustrates that process and the computations involved.

Objectives

In this lab, you'll learn how to:

- compute OLS estimates;
- calculate fitted values and residuals;
- compute the error variance estimate;
- compute the variance-covariance matrix of $\hat{\beta}$, which quantifies the variability of model estimates;
- compute standard errors for each model estimate;
- compute the proportion of variation captured by a linear model.

Throughout you'll use simple visualizations to help make the connection between fitted models and the aspects of a dataset that model features describe.

Data: fertility rates

By way of data, you'll work with country indicators, total fertility rates, and gender indicators for a selection of countries in 2018, and explore the decline in fertility rates associated with developed nations. Data from the U.S. 2020 census indicated significant population growth decline in the United States. If the topic interests you, you can read more about perspectives and existing data in this Our World in Data article.

The data are stored in separate `.csv` files imported below:

In [ ]:
```
fertility = pd.read_csv('data/fertility.csv')
country = pd.read_csv('data/country-indicators.csv')
gender = pd.read_csv('data/gender-data.csv')
```

The variables you'll work with in this portion are the following:

Dataset	Name	Variable	Units
`fertility`	`fertility_total`	National fertility rate	Average number of children per woman

Dataset	Name	Variable	Units
`country`	`hdi`	Human development index	Index between 0 and 1 (0 is lowest, 1 is highest)
`gender`	`edu_expected_yrs_f`	Expected years of education for adult women	Years

Because the variables of interest are stored in three separate dataframes, you'll first need to extract them and merge by country.

In [ ]:
```
# slice variables of interest
fertility_sub = fertility.loc[:, ['Country', 'fertility_total']]
gender_sub = gender.loc[:, ['educ_expected_yrs_f', 'Country']]
country_sub = country.loc[:, ['Country', 'hdi']]

# merge variables of interest
reg_data = pd.merge(
    fertility_sub,
    gender_sub,
    on = 'Country',
    how = 'inner'
).merge(
    country_sub,
    on = 'Country',
    how = 'left'
).set_index('Country').dropna()

# preview
reg_data.head(4)
```

We'll treat the fertility rates as our variable of interest.

Exploratory analysis

A preliminary step in regression analysis is typically data exploration through scatterplots. The objective of exploratory analysis in this context is to identify an approximately linear relationship to model.

Question 1: Education and fertility rate

Construct a scatterplot of total fertility against expected years of education for women. Label the axes 'Fertility rate' and 'Expected years of education for women'. Store this plot as `scatter_educ` and display the graphic.

(*Remark*: be sure to include `scale = alt.Scale(zero = False)` in the axis specification so that your plot does not have extra whitespace.)

In [ ]:
```
...
```

This figure shows a clear negative association between fertility rate and women's educational attainment, *and* that the relationship is roughly linear. Next, check whether HDI seems to be related to fertility rate.

Question 2: HDI and fertility rate

Now construct a scatterplot comparing fertility rate with HDI. Make sure you choose appropriate labels for your axes and plot. Store this plot as `scatter_hdi` and display the graphic.

In [ ]:
```
...
```

This figure shows a negative relationship between fertility rate and HDI; it may not be exactly linear, but a line should provide a decent approximation. So, the plots suggest that a linear regression model in one or both explanatory variables is reasonable.

Simple linear regression

To start you'll fit a simple linear model regressing fertility on education.

First we'll need to store the quantities -- the response and explanatory variables -- needed for model fitting in the proper format. Recall that the linear model in matrix form is:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}}_{\beta} + \underbrace{\begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}}_{\epsilon}$$

Notice that the explanatory variable matrix $\mathbf{X}$ includes a column of ones for the intercept. So the quantities needed are:

- $\mathbf{y}$, a one-dimensional array of the total fertility rates for each country; and
- $\mathbf{X}$, a two-dimensional array with a column of ones (intercept) and a column of the expected years of education for women (explanatory variable).

The cell below constructs these arrays as pandas objects.

In [ ]:
```
# retrieve response
y = reg_data.fertility_total

# construct explanatory variable matrix
x = sm.tools.add_constant(reg_data.educ_expected_yrs_f)

# print first five rows of x
x.head()
```

Estimation

'Fitting' a model refers to computing estimates; `statsmodels.OLS()` will fit a linear regression model based on the response vector and explanatory variable matrix. Note that the model structure is implicit -- `OLS` will fit $y = X\beta + \epsilon$ no matter what, so you need to be sure you have arranged X and y correctly to fit the model that you intend.

In [ ]:
```
# fit model
slr = sm.OLS(endog = y, exog = x)
```

This returns an object of a distinct model class specific to `OLS`:

In [ ]:
```
type(slr)
```

Associated with the class are various attributes and methods. From the model instance, `.fit()` retrieves the model results:

In [ ]:
```
type(slr.fit())
```

Note, however, that `slr.fit()` will not produce any interesting output:

In [ ]:
```
slr.fit()
```

What the `.fit()` method does is create a results object that contains parameter estimates and other quantities we might want to retrieve.

In [ ]:
```
rslt = slr.fit()
```

The coeffient estimates $\hat{\beta}_0, \hat{\beta}_1$ are:

In [ ]:
```
rslt.params
```

The error variance estimate $\hat{\sigma}^2$ is:

In [ ]:
```
rslt.scale
```

It was noted in lecture that the variances and covariances of $\hat{\beta}_0, \hat{\beta}_1$ are given by the matrix:

$$\sigma^2(\mathbf{X}'\mathbf{X})^{-1} = \begin{bmatrix} \text{var}\hat{\beta}_0 & \text{cov}\left(\hat{\beta}_0, \hat{\beta}_1\right) \\ \text{cov}\left(\hat{\beta}_1, \hat{\beta}_0\right) & \text{var}\hat{\beta}_1 \end{bmatrix}$$

So we can *estimate* these quantities, which quantify the variation and covariation of the estimated coefficients, by plugging in the estimated error variance and computing $\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$. This estimate is:

In [ ]:
```
rslt.cov_params()
```

Standard errors for the coefficient estimates are obtained from the diagonal entries. We might create a nice summary of all the estimates as follows:

In [ ]:
```
coef_tbl = pd.DataFrame({'estimate': rslt.params.values,
                         'standard error': np.sqrt(rslt.cov_params().values.diagonal())},
                        index = x.columns)
coef_tbl.loc['error variance', 'estimate'] = rslt.scale

coef_tbl
```

Lastly, a standard metric often reported with linear models is the R^2 score, which is interpreted as the proportion of variation in the response captured by the model.

In [ ]:
```
# compute R-squared
rslt.rsquared
```

So, the expected years of education for women in a country explains 72% of variability in fertility rates, and furthermore, according to the fitted model:

- for a country in which women are entirely uneducated, the estimated mean fertility rate is 7.5 children on average by the end of a woman's reproductive period
- each additional year of education for women is associated with a decrease in a country's fertility rate of an estimated 0.43
- after accounting for women's education levels, fertility rates vary by a standard deviation of $0.66 = \sqrt{0.438}$ across countries

Question 3: center the explanatory variable

Note that *no* countries report an expected zero years of education for women, so the meaning of the intercept is artificial. As we saw in lecture, centering the explanatory variable can improve interpretability of the intercept. Center the expected years of education for women and refit the model by following the steps outlined below. Display the coefficient estimates and standard errors.

In [ ]:
```
# center the education column by subtracting its mean from each value
educ_ctr = ...

# reconstruct the explanatory variable matrix
x_ctr = ...

# fit new model
slr_ctr = ...
rslt_ctr = ...

# arrange estimates and se's in a dataframe and display
...
```

In [ ]:
```
grader.check("q3")
```

Fitted values and residuals

The **fitted value** for y_i is the value along the line specified by the model that corresponds to the matching explanatory variable x_i. In other words:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

These can be obtained directly from `rslt`:

In [ ]:
```
# fitted values
rslt.fittedvalues
```

The result is an array with length matching the number of rows in `x`; note the index for the pandas series -- the fitted values are returned in the same order as the observations used to fit the model.

In [ ]:
```
(rslt.fittedvalues.index == x.index).all()
```

Recall that model **residuals** are the difference between observed and fitted values:

$$e_i = y_i - \hat{y}_i$$

These are similarly retrievable as an attribute of the regression results:

In [ ]:
```
# residuals
rslt.resid
```

Note again that these are returned in the same order as the original observations.

Question 4: calculations 'by hand'

Calculate the fitted values and residuals manually. Store the results as arrays `fitted_manual` and `resid_manual`, respectively.

Hint: use matrix-vector multiplication.

In [ ]:
```
fitted_manual = ...
resid_manual = ...
```

In [ ]:
```
grader.check("q4")
```

It is often convenient to add the fitted values and residuals as new columns in `reg_data`:

In [ ]:
```
# append fitted values and residuals
reg_data['fitted_slr'] = rslt.fittedvalues
reg_data['resid_slr'] = rslt.resid
```

```
reg_data.head(3)
```

We can use this augmented dataframe to visualize the deterministic part of the model:

In [ ]:
```
# construct line plot
slr_line = alt.Chart(reg_data).mark_line().encode(
    x = 'educ_expected_yrs_f',
    y = 'fitted_slr'
)

# layer
scatter_educ + slr_line
```

To obtain uncertainty bands about the estimated mean, we'll compute predictions at each observed value using `.get_prediction()` -- this method by default returns standard errors associated with each prediction.

In [ ]:
```
preds = rslt.get_prediction(x)
```

The predictions are stored as `.predicted_mean`. Since we computed predictions at the observed values, the predictions should match the fitted values:

In [ ]:
```
(preds.predicted_mean == rslt.fittedvalues).all()
```

Standard errors are stored as `.se_mean`. Uncertainty bands are typically drawn $2SE$ in either direction from the fitted values; so we'll append those values to the original data.

In [ ]:
```
reg_data['lwr_mean'] = preds.predicted_mean - 2*preds.se_mean
reg_data['upr_mean'] = preds.predicted_mean + 2*preds.se_mean

reg_data.head()
```

We can use these to shade in the area between the lower and upper limits.

In [ ]:
```
band = alt.Chart(reg_data).mark_area(opacity = 0.2).encode(
    x = 'educ_expected_yrs_f',
    y = 'lwr_mean',
    y2 = 'upr_mean'
)

# layer
scatter_educ + slr_line + band
```

As discussed in lecture, we can also compute and display uncertainty bounds for predicted observations (rather than the mean). These will be wider, because there is more uncertainty associated with predicting observations compared with estimating the mean.

Question 5: prediction intervals

The standard error for *predictions* is stored with the output of `.get_prediction()` as the attribute `.se_obs` -- standard error for observations. Use this and follow the example above to compute 95% uncertainty bounds for the observations. Add the lower and upper bounds as new columns of `reg_data` named `lwr_obs` and `upr_obs`, respectively. Construct a plot showing data scatter, the model predictions, and prediction uncertainty bands.

In [ ]:
```
# compute prediction uncertainty bounds
reg_data['lwr_obs'] = ...
reg_data['upr_obs'] = ...

# construct plot showing prediction uncertainty
...
```

In [ ]:
```
grader.check("q5")
```

Recall that the interpretation of the prediction band is that 95% of the time, the band will cover the observed values.

Question 6: coverage

What proportion of observed values are within the prediction bands? Compute and store this value as `coverage_prop`.

In [ ]:
```
coverage_prop = ...
```

In [ ]:
```
grader.check("q6")
```

Multiple linear regression

Now let's consider adding the human development factor to the model. First let's investigate the *univariate* relationship between HDI and fertility rate.

A scatterplot is shown below with a regression line overlaid. The relationship perhaps isn't perfectly linear, but a line should provide a decent approximation.

In [ ]:
```
base = alt.Chart(reg_data).mark_circle().encode(
    x = alt.X('hdi', title = 'Human development index', scale = alt.Scale(zero = Fa
    y = alt.Y('fertility_total', title = 'Fertility rate', scale = alt.Scale(zero =
)

base + base.transform_regression(on = 'fertility_total', regression = 'hdi').mark_l
```

Question 7: fit a model with HDI only

Fit the model plotted above. Display the coefficient estimates, standard errors, and R^2 statistic.

In [ ]:
```
# fit model
x = ...
slr = ...
rslt = ...

# construct coefficient table
coef_tbl = pd.DataFrame({
    ...
    ...
    },
    index = ...
)

coef_tbl.loc['error variance', 'estimate'] = ...

# display table and print r squared
...
```

You should have observed that this model *also* explains about 70% of variance in fertility rates. So it seems like an equally good predictor of fertility rates. However, HDI is highly correlated with women's education:

In [ ]:
```
reg_data.corr().loc['hdi', 'educ_expected_yrs_f']
```

So what do you think will happen if we fit a model with both explanatory variables? Will fertility rate have a stronger association with one or the other? Will the coefficient estimates also be highly correlated? Take a moment to consider this and come up with a hypothesis.

The model is fit *exactly* the same way as the SLR models -- all we need change is the explanatory variable matrix. Instead of grabbing one column from the data, we now grab two:

In [ ]:
```
# construct explanatory variable matrix
x = sm.tools.add_constant(reg_data.loc[:, ['hdi', 'educ_expected_yrs_f']])

# fit model
mlr = sm.OLS(endog = y, exog = x)

# store results
rslt = mlr.fit()
```

The coefficient table is shown below:

In [ ]:
```
coef_tbl = pd.DataFrame({
    'estimate': rslt.params.values,
    'standard error': np.sqrt(rslt.cov_params().values.diagonal())
    },
    index = x.columns
```

```
)

coef_tbl.loc['error variance', 'estimate'] = rslt.scale

coef_tbl
```

The association with HDI is not as strong as in the simple linear model -- note the coefficient estimate here is about -4.1, compared with about -7.0 if education is not included.

Similarly, the association with education is not as strong as in the simple linear model with only education -- if HDI is not included in the model, the coefficient estimate is about -0.43, whereas with both variables the estimate is about -0.20.

So both associations are weaker when both terms are included in the model. Further, the estimates are strongly correlated:

In [ ]:
```
vcov = rslt.cov_params()
stderr = np.sqrt(vcov.values.diagonal())
corr_mx = np.diag(1/stderr).dot(vcov).dot(np.diag(1/stderr))

print('correlation of hdi and educ coefficient estimates: ', corr_mx[1, 2])
```

The multiple linear regression model captures a little bit more variance than either simple linear regression model individually:

In [ ]:
```
rslt.rsquared
```

The MLR model doesn't add much value in terms of fit, so if that is our only concern we might prefer one of the SLR models. However, the presence of additional predictors changes the parameter interpretation -- in the MLR model, the coefficients give the estimated changes in mean fertility rate associated with changes in each explanatory variable *after accounting for the other explanatory variable*. This is one way of understanding why the estimates change so much in the presence of additional explanatory variables -- the association between, *e.g.*, HDI and fertility, is different than the association between HDI and fertility after adjusting for women's expected education.

More broadly, these data are definitely *not* a representative sample of any particular population of nations -- the countries (observational units) are conveniently chosen based on which countries reported data. So there is no scope of inference here, for any of the models we've fit.

Although we can't claim that, for example, 'the mean fertility rate decreases with education at a rate of 0.2 children per woman per expected year of education after accounting for development status', we *can* say '*among the countries reporting data*, the mean fertility rate decreases with education at a rate of 0.2 children per woman per expected year of education after accounting for development status'. This is a nice example of how a model might be used in a descriptive capacity.

Submission

1. Save the notebook.
2. Restart the kernel and run all cells. (**CAUTION**: if your notebook is not saved, you will lose your work.)
3. Carefully look through your notebook and verify that all computations execute correctly and all graphics are displayed clearly. You should see **no errors**; if there are any errors, make sure to correct them before you submit the notebook.
4. Download the notebook as an `.ipynb` file. This is your backup copy.
5. Export the notebook as PDF and upload to Gradescope.

To double-check your work, the cell below will rerun all of the autograder tests.

In [ ]:
```
grader.check_all()
```