In [1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab6-regression.ipynb")
```

In [2]:
```
import numpy as np
import pandas as pd
import altair as alt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import add_dummy_feature

alt.data_transformers.disable_max_rows()
```

Out[2]:
```
DataTransformerRegistry.enable('default')
```

Lab 6: Regression

This lab covers the nuts and bolts of fitting linear models using the `sklearn.linear_model` module. The linear model expresses a response variable, y, as a linear function of $p-1$ explanatory variables $x_1, \dots, x_{p-1}$ 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 $i = 1, \dots, n$ so that the model describes a dataset comprising n values of each variable:

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

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:

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

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

- The ordinary least squares 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 use the `sklearn.linear_model` module to:

- compute OLS estimates;
- calculate fitted values and residuals;
- compute the error variance estimate.

In addition, you'll see how to calculate:

- the variance-covariance matrix of $\hat{\beta}$, which quantifies the variability of model estimates;
- standard errors for each model estimate;
- 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. The lab activity proceeds through the following sections:

1. Exploratory plots for regression analysis
2. Simple linear regression (single explanatory variable)
    - Model fitting
    - Uncertainty quantification
    - Model visualization
3. Multiple linear regression (several explanatory variables)
    - Data preprocessing: encoding categorical variables
    - Interaction terms
    - Model fitting
    - Uncertainty quantification
    - Model visualization

Data: fertility rates and meausres of development

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. This has been in the news lately due to preliminary data from the U.S. 2020 census indicating significant [population growth decline in the United States (https://www.washingtonpost.com/dc-md-va/interactive/2021/2020-census-us-population-results/)](https://www.washingtonpost.com/dc-md-va/interactive/2021/2020-census-us-population-results/). If the topic interests you, you can read more about perspectives and existing data in this [Our World in Data article (https://ourworldindata.org/fertility-rate)](https://ourworldindata.org/fertility-rate).

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

In [3]:
```
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
`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 [4]:
```
# 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)
```

Out[4]:

Country	fertility_total	educ_expected_yrs_f	hdi
Afghanistan	4.473	6.795722	0.509
Albania	1.617	13.201755	0.792
Algeria	3.023	12.108990	0.746
Angola	5.519	6.973901	0.582

0. Exploratory plots

A preliminary step in regression analysis is typically data exploration through scatterplots. This relies on skills you've already developed -- you may find it helpful to refer to Lab 3 (visualization) throughout this section.

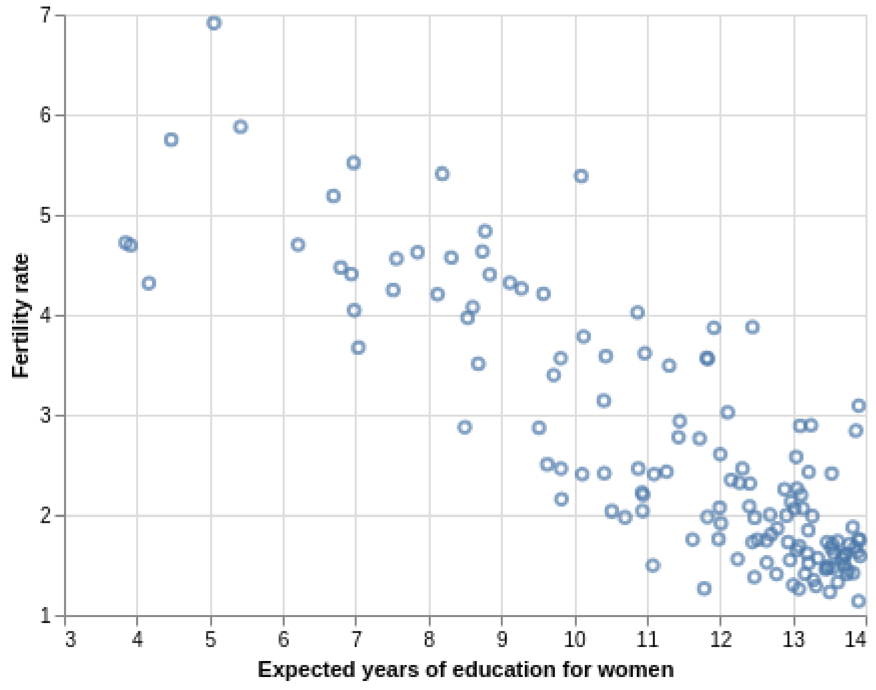
Q0 (a) Simple scatterplot

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 `simple_scatter` 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 [5]:
```python
#solution
simple_scatter = alt.Chart(reg_data).mark_point().encode(
    x = alt.X('educ_expected_yrs_f',
              scale = alt.Scale(zero = False),
              title = 'Expected years of education for women'),
    y = alt.Y('fertility_total',
              scale = alt.Scale(zero = False),
              title = 'Fertility rate')
)
simple_scatter
```

Out[5]:



The cell below coerces the human development index to a high-med-low factor (categorical variable) using `pd.qcut(...)`. The cutoffs for the factor levels are determined by sample quantiles.

In [6]:
```python
# labels for the factor
hdi_labels = ['1_low', '2_medium', '3_high']

# cut hdi based on quantiles and store as new variable
reg_data['hdi_fac'] = pd.qcut(reg_data.hdi,
                              q = 3,
                              labels = hdi_labels)

# preview
reg_data.head(3)
```

Out[6]:

	fertility_total	educ_expected_yrs_f	hdi	hdi_fac
Country				
Afghanistan	4.473	6.795722	0.509	1_low
Albania	1.617	13.201755	0.792	2_medium
Algeria	3.023	12.108990	0.746	2_medium

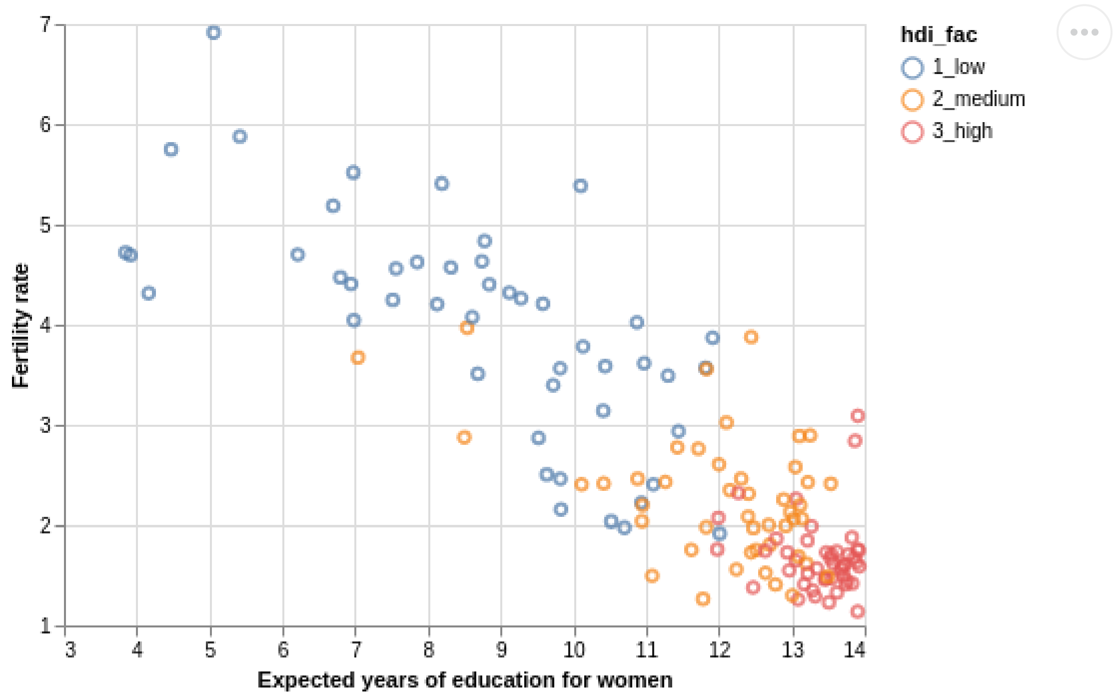
Q0 (b) Scatter by HDI level

(i) Add HDI level

Modify your plot from Q0 (a) so that points are colored according to the corresponding country's level of human development. Make sure you choose appropriate labels for your axes and plot. Store this plot as `scatter` and display the graphic.

In [7]:
```
scatter = simple_scatter.encode(
    color = alt.Color('hdi_fac')
)
scatter
```

Out[7]:



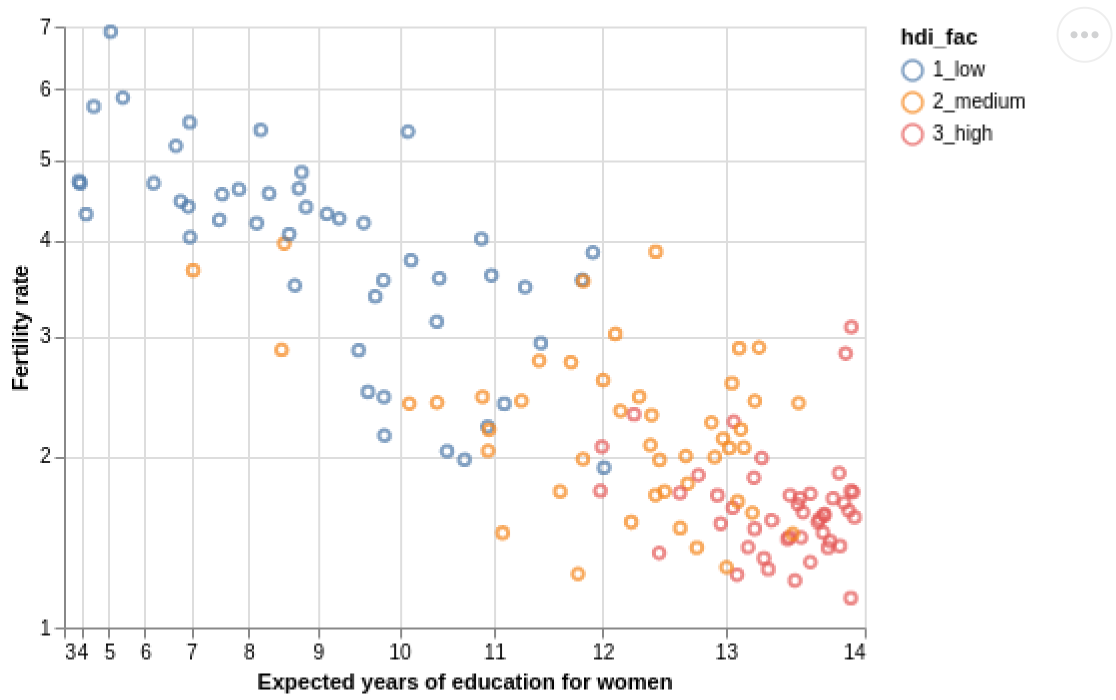
(ii) Adjust axis scales

Notice that the scatter is clumped together densely in the lower right corner. To better see what's happening in that region, modify your codes for constructing `scatter` so that your axis specifications for both axes include `alt.Scale(zero = False, type = 'pow', exponent = ...)`. Make sure you choose appropriate labels for your axes and plot.

This will display the axes on a power-transformed scale with an exponent of your choosing. Use an exponent around 2.5 for the education axis, and an exponent around 1/3 for the fertility axis. Store the plot as `scatter_scaled` and display the result.

In [8]:
```
scatter_scaled = scatter.encode(
    x = alt.X('educ_expected_yrs_f',
              scale = alt.Scale(zero = False, type = 'pow', exponent = 2.5),
              title = 'Expected years of education for women'
             ),
    y = alt.Y('fertility_total',
              scale = alt.Scale(zero = False, type = 'pow', exponent = 1/3),
              title = 'Fertility rate'
             )
)
scatter_scaled
```

Out[8]:

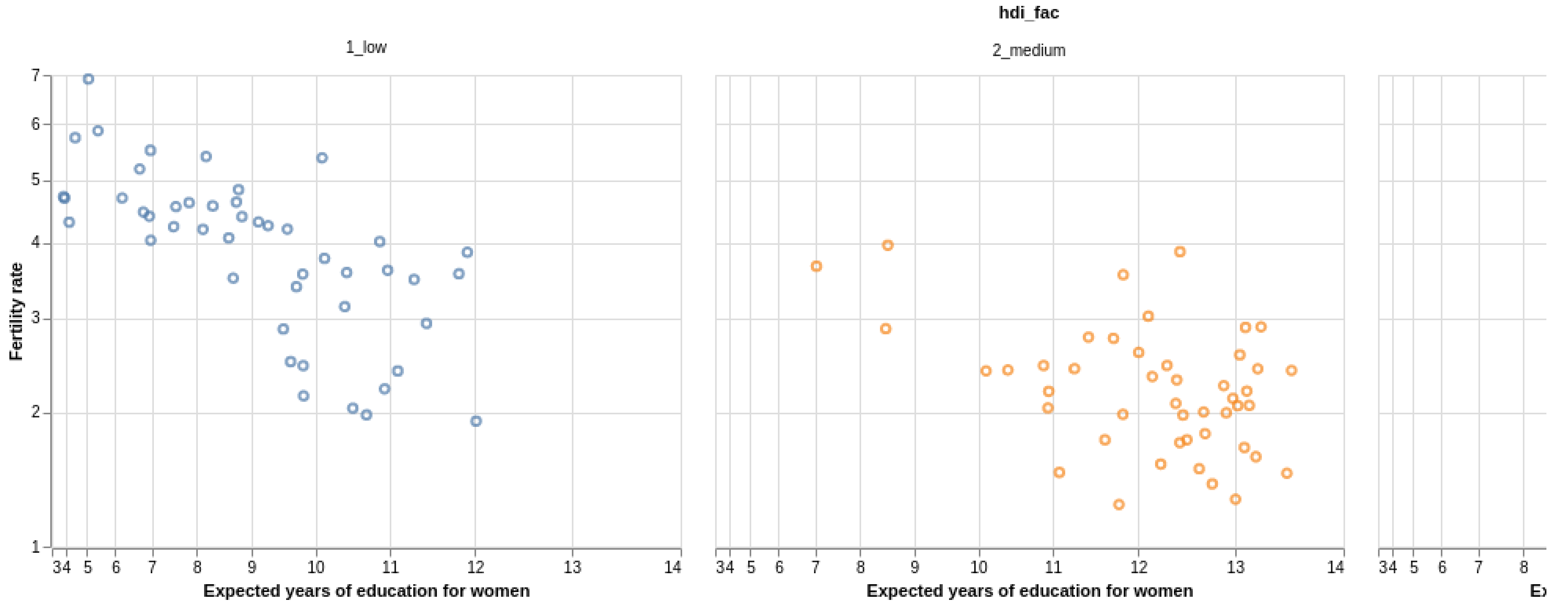


Q0 (c) Interpret your graphic

Does the negative association (downward trend in the scatter) seem to hold steady among countries at each level of human development? Try to imagine drawing a line to fit the points of each color, and ask yourself whether the slopes would be the same.

i. Facet by human development (`hdi_fac` level) and store the result in `faceted_hdi` so that you can see the pattern for each development level apart from the other levels.

In [9]:
```
# facet by hdi level
faceted_hdi = scatter_scaled.facet('hdi_fac')
faceted_hdi
```
Out[9]:



Does the negative association (downward trend in the scatter) seem to hold steady among countries at each level of human development?

Yes.

Examining individual points

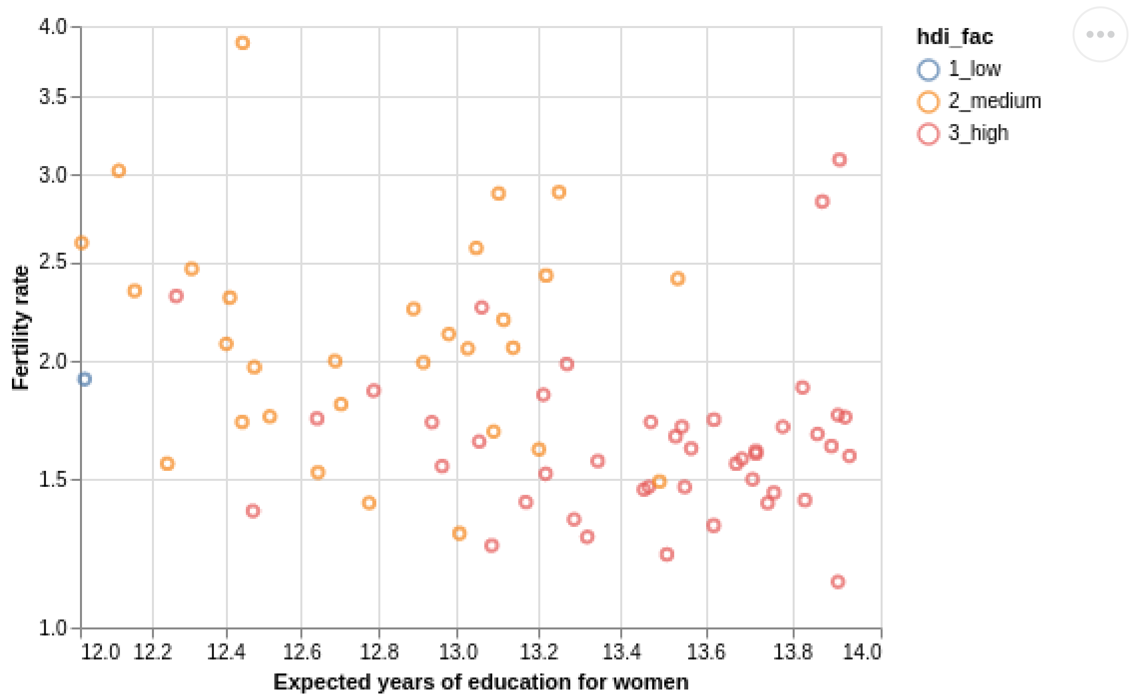
Often it can be helpful to graphically examine individual points. Especially so if you notice outliers. In this dataset there aren't any apparent outliers, but there are a few points that seem a little farther from the scatter than others, so we can practice on those.

Let's zoom in on the lower-right region of the scatterplot; the cell below applies an Altair filter transform to show only data for which expected education exceeds 12.

Please be sure you've created `scatter_scaled` in Q0 (b) (ii) before running this cell.

In [10]:
```
# zoom in
scatter_highed = scatter_scaled.transform_filter(
    alt.FieldGTPredicate(field = 'educ_expected_yrs_f', gt = 12)
)

scatter_highed
```
Out[10]:



Notice that there are a few points that stand apart from the main scatter. Which countries do those points represent? It's not to tricky to pick them out by filtering and taking maxima or minima. For example, the country with the lowest fertility rate is:

In [11]:
```
reg_data[reg_data.educ_expected_yrs_f > 12].fertility_total.idxmin()
```
Out[11]:
```
'Singapore'
```

Q0 (e) Labeling specific countries

Here you'll look at a few more specific countries, and then label them on the plot.

(i) Find the country among the high-HDI nations with the highest total fertility and store it as `high_high_fert`.

In [12]:
```
high_high_fert = reg_data[
    (reg_data.hdi_fac == '3_high') & (reg_data.educ_expected_yrs_f > 12)
].fertility_total.idxmax()

print(high_high_fert)
```

```
Israel
```

(ii) Find the country among the medium-HDI nations with the highest total fertility and store it as `med_high_fert`.

In [13]:
```
med_high_fert = reg_data[
    (reg_data.hdi_fac == '2_medium') & (reg_data.educ_expected_yrs_f > 12)
].fertility_total.idxmax()

print(med_high_fert)
```

```
Samoa
```

(iii) Identify one other country that calls your attention.

In [14]:
```
#BEGIN SOLUTION NO PROMPT
#potential solution
reg_data[(reg_data.hdi_fac == '1_low')  & (reg_data.educ_expected_yrs_f > 12)].fertility_total.idxmax()
#END SOLUTION
```

Out[14]:
```
'Nepal'
```

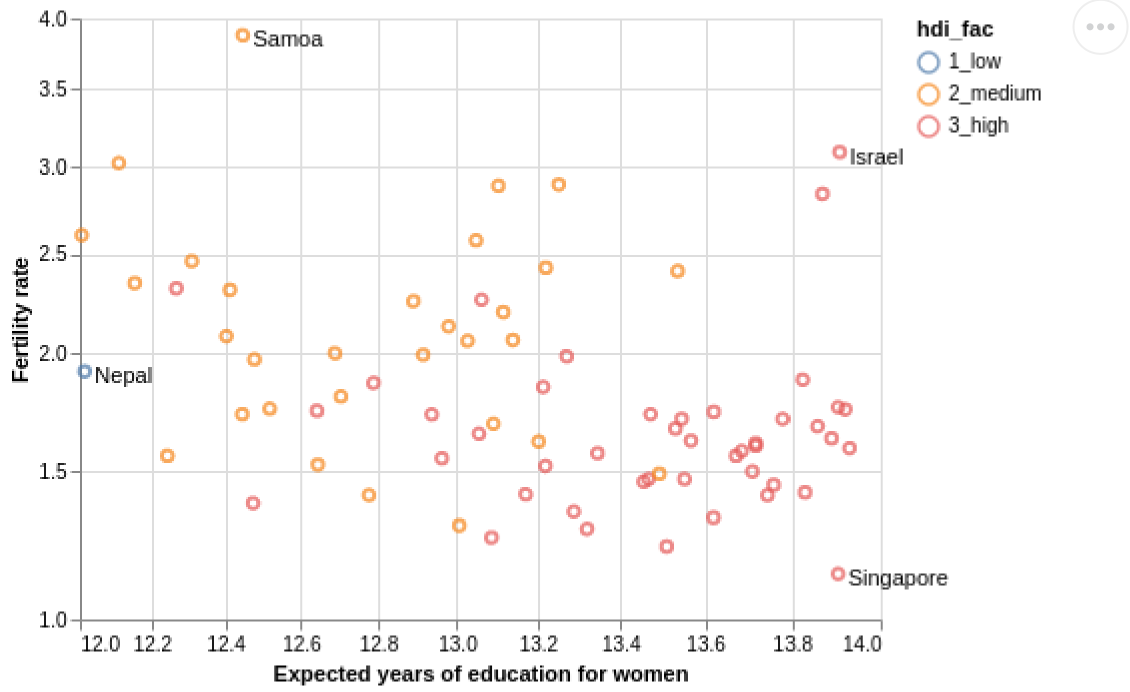
(iv) Modify the cell below to label the three countries identified above.

In [15]:
```
# slice rows for identified countries
point_labels = reg_data.loc[['Singapore', 'Israel', 'Samoa', 'Nepal']].reset_index() # edit here

# plot text labels
labels = alt.Chart(point_labels).mark_text(align = 'left', dx = 5, dy = 3).encode(
    x = 'educ_expected_yrs_f',
    y = 'fertility_total',
    text = 'Country'
)

# layer over scatter
scatter_highed + labels
```

Out[15]:



1. Simple linear regression

In this part 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 prepares the explanatory variable matrix $\mathbf{X}$ and response vector $\mathbf{y}$ in the following steps:

1. Store the total fertility rates as an array named `y`.
2. Slice the `educ_expected_yrs_f` variable from the `reg_data` dataframe, resulting in a dataframe with one column stored as `x_slr_df`.
3. Append a column of ones to `x_slr_df` using `add_dummy_feature`, resulting in a 139-by-2 array stored as `x_slr`.

In [16]:
```
# store response variable as array
y = reg_data.fertility_total

# store explanatory variable
x_slr_df = reg_data.loc[:, ['educ_expected_yrs_f']]

# add intercept column
x_slr = add_dummy_feature(x_slr_df, value = 1)

# print first five rows
x_slr[0:5, :]
```

Out[16]:
```
array([[ 1.        ,  6.79572248],
       [ 1.        , 13.20175457],
       [ 1.        , 12.10898972],
       [ 1.        ,  6.97390127],
       [ 1.        , 12.91444111]])
```

Model fitting

'Fitting' a model refers to computing estimates; it is typical practice to report estimated parameters and standard errors, but first we'll focus on computing the estimates. This is done using the function `LinearRegression()` in the `sklearn.linear_model` module. The syntax is superficially somewhat similar to using `PCA(...)` in the previous lab (also from an `sklearn` module): we'll configure `LinearRegression(...)` as `slr`, and then fit the model using the `fit(...)` method.

In [17]:
```
# configure module
slr = LinearRegression(fit_intercept = False)

# fit model
slr.fit(x_slr, y)
```

Out[17]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

Notice the argument `fit_intercept = False` -- this configures `slr` *not* to fit an intercept, which is done here since the explanatory variable matrix already includes a column of ones for the intercept term.

Q1 (a) Retrieve estimated coefficients

After the `fit(...)` method is applied, the coefficient estimates are stored as the `.coef_` attribute of `slr`.

(i) Store the coefficient estimates in `coef` and print them.

In [18]:
```
#store the coefficient estimates
coef = slr.coef_

# print coefficient estimates
print(coef)
```

```
[ 7.51142307 -0.42747211]
```

In [19]:
```
grader.check("q1_a_i")
```

Out[19]: **q1_a_i** passed!

The order in which the `.coef_` attribute stores the estimates matches the order of the columns of $\mathbf{X}$.

(ii) Which is which?

Identify the intercept and identify the slope. Replace the dots below.

- Intercept estimate: $\hat{\beta}_0 = \dots$
- Slope estimate: $\hat{\beta}_1 = \dots$

(*Hint*: refer back to the plot; only one of the two estimates could possibly be the slope.)

In [20]:
```
b0 = coef[0]
b1 = coef[1]
```

In [21]:
```
grader.check("q1_a_ii")
```

Out[21]: **q1_a_ii** passed!

(iii) Interpret the slope coefficient.

Fill in the blanks below.

Among the countries in the sample, a one-year increase in expected years of education for women is associated with a 7.5 decrease in the average number of children per woman.

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 by passing the explanatory variable matrix to the `.predict(...)` method. In effect, this returns predictions on the observed data.

In [22]:
```
# fitted values
fitted_slr = slr.predict(x_slr)

fitted_slr[0:5]
```

Out[22]:
```
array([4.60644126, 1.86804122, 2.33516772, 4.5302748 , 1.99085971])
```

The result is an array with length matching the number of rows in `x_slr`.

In [23]:
```
# dimensions
fitted_slr.shape
```

Out[23]:
```
(139,)
```

Q1 (b) Compute the residuals

The 'residuals' -- what's left over -- are the differences between the fitted and observed values of the response:

$$e_i = y_i - \hat{y}_i \qquad i\text{th residual}$$

Use `fitted_slr` and `y` to compute an array of the residuals. Store the result as `resid_slr`.

In [24]:
```
# residuals
resid_slr = y - fitted_slr

#print
resid_slr
```

Out[24]:
```
Country
Afghanistan           -0.133441
Albania               -0.251041
Algeria                0.687832
Angola                 0.988725
Antigua and Barbuda    0.003140
                         ...
Uruguay               -0.204590
Uzbekistan             0.224314
Vanuatu                0.602464
Zambia                 0.859878
Zimbabwe               0.795048
Name: fertility_total, Length: 139, dtype: float64
```

```
In [25]: grader.check("q1_b")
```

Out[25]: **q1_b** passed!

Q1 (c) Append fitted values and residuals to the data

In order to keep them handy, add the fitted values and residuals as new columns in `reg_data` with the names `fitted_slr` and `resid_slr`. Print the first few rows of `reg_data` after adding the new variables.

```
In [26]: # append
reg_data['fitted_slr'] = fitted_slr
reg_data['resid_slr'] = resid_slr

# print
reg_data.head(5)
```

Out[26]:

	fertility_total	educ_expected_yrs_f	hdi	hdi_fac	fitted_slr	resid_slr
Country						
Afghanistan	4.473	6.795722	0.509	1_low	4.606441	-0.133441
Albania	1.617	13.201755	0.792	2_medium	1.868041	-0.251041
Algeria	3.023	12.108990	0.746	2_medium	2.335168	0.687832
Angola	5.519	6.973901	0.582	1_low	4.530275	0.988725
Antigua and Barbuda	1.994	12.914441	0.772	2_medium	1.990860	0.003140

```
In [27]: grader.check("q1_c")
```

Out[27]: **q1_c** passed!

There's one more estimate to compute: the error variance, σ^2! This can be estimated based on the residual variance:

$$\hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^{n} e_i^2 = \frac{n-1}{n-p} S_e^2$$

In this expression, e_i are the residuals, n and p are the dimensions of $\mathbf{X}$, and S_e^2 is the residual (sample) variance.

```
In [28]: # store n (number of observations) and p (number of betas)
n, p = x_slr.shape

# compute estimate of error variance
sigma2_hat = ((n - 1)/(n - p)) * resid_slr.var()
```

Uncertainty quantification

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:

$$\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$$

```
In [29]: # compute X'X
xtx_slr = x_slr.transpose().dot(x_slr)

# compute matrix of parameter variances/covariances: estimated error variance x (X'X)^{-1}
slrcoef_vcov = np.linalg.inv(xtx_slr) * sigma2_hat

# print
slrcoef_vcov
```

```
Out[29]: array([[ 0.0688336 , -0.0057818 ],
       [-0.0057818 ,  0.00050894]])
```

Q1 (d) Standard errors

The *standard errors* for the estimates $\hat{\beta}_0, \hat{\beta}_1$ are the square roots of their estimated variances. Compute these by retrieving the diagonal elements of `slrcoef_vcov` (the estimated variances) and taking square roots.

Store the result (an array with two values) as `slrcoef_se` and print the array.

(*Hint*: use `.diagonal()` from numpy.)

In [30]:
```
# take square root of matrix diagonals to get standard errors
slrcoef_se = np.sqrt(slrcoef_vcov.diagonal())

# print
slrcoef_se
```

Out[30]: `array([0.26236158, 0.02255976])`

In [31]:
```
grader.check("q1_d")
```

Out[31]: **q1_d** passed!

We can now report the results of model fitting in an organized fasion: we'll make a dataframe `slrcoef_table` with two columns, 'coefficient estimate' and 'standard error', and index the rows of the dataframe by the coefficient names (intercept and education).

In [32]:
```
# create table
slrcoef_table = pd.DataFrame(
    data = {'coefficient estimate': slr.coef_, 'standard error': slrcoef_se},
    index= ['intercept', 'education']
)

# print
slrcoef_table
```

Out[32]:

	coefficient estimate	standard error
intercept	7.511423	0.262362
education	-0.427472	0.022560

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. It is straightforward to compute using the `r2_score(...)` function from `sklearn.metrics` :

In [33]:
```
# compute R-squared
r2_score(reg_data.fertility_total, reg_data.fitted_slr)
```

Out[33]: `0.723814308777671`

But just so you have a sense of what it is, a direct calculation is shown below:

In [34]:
```
# R-squared 'by hand'
(y.var() - resid_slr.var())/y.var()
```

Out[34]: `0.7238143087776711`

The metric is simply the difference between the raw variation in the response and residual variation, as a proportion of the variation in the response. If the model fits well, the residual variation will be small, in which case this proportion will be closer to 1.

Q1 (e) Visualize!

Now that you've reported the numerical results of model fitting, have a direct look at the results relative to the data scatter.

(i) Construct a line plot of the fitted values against expected years of education.

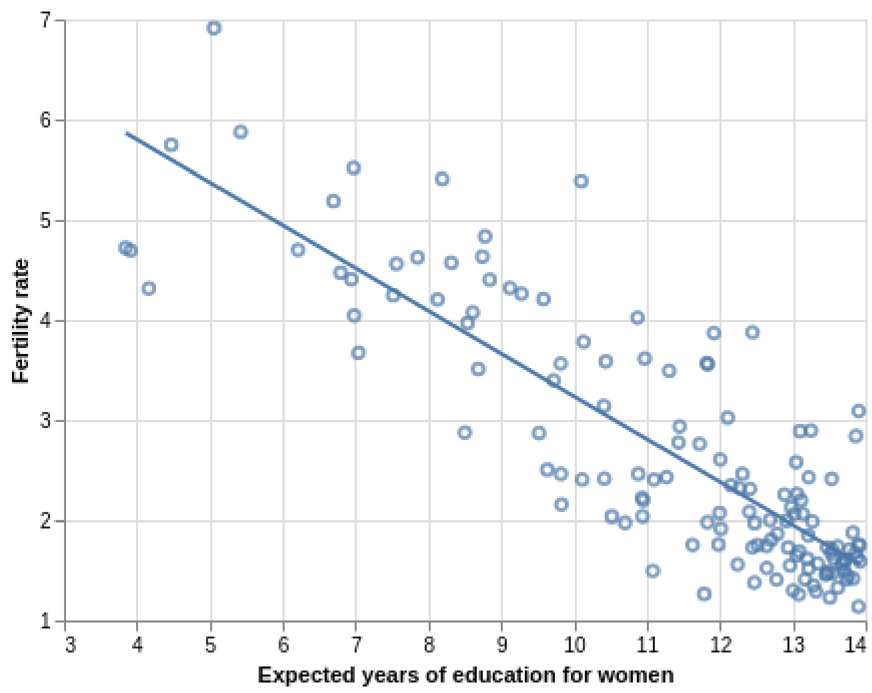
Remember that the fitted values are stored as `fitted_slr` in the dataframe `reg_data` containing the actual data.

Construct the line as `slr_line` and layer this on top of your plot `simple_scatter` from part 0 above.

In [35]:
```python
# construct line plot
slr_line = alt.Chart(reg_data).mark_line().encode(
    x = alt.X('educ_expected_yrs_f', scale = alt.Scale(zero = False)),
    y = alt.Y('fitted_slr', scale = alt.Scale(zero = False))
)

# layer
simple_scatter + slr_line
```

Out[35]:



(ii) Comment on the plot.

How well does the model seem to describe the data overall?

It decribes the data pretty well, a large part of points gather around the line.

---

2. Multiple linear regression

Now let's consider adding the human development factor -- a categorical variable -- to the model. The factor column `hdi_fac` can't be input directly as a variable into the explanatory variable matrix $\mathbf{X}$, because it doesn't make sense to multiply a coefficient (number) by a factor label (character string), so we'll need to do a little preprocessing to incorporate the factor into a multiple regression model.

Encoding categorical variables

Categorical variables must be *encoded* in the explanatory variable matrix by *indicator functions*; each level of the category will have a corresponding coefficient. In this case, the factor (HDI) had three levels (low, medium, and high), so we'll construct a matrix $\mathbf{X}$ with entries

$$\left[\begin{array}{cccc} 1 & x_1 & I(hdi_1 = med) & I(hdi_1 = high) \\ 1 & x_2 & I(hdi_2 = med) & I(hdi_2 = high) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & I(hdi_n = med) & I(hdi_n = high) \end{array}\right]$$

where

- $I(hdi_i = med)$ is 1 if the ith HDI level is medium, and 0 otherwise and
- $I(hdi_i = high)$ is 1 if the ith HDI level is high, and 0 otherwise.

The low level of the HDI factor is not represented explicitly, but if $I(hdi_i = med) = I(hdi_i = high) = 0$, then $hdi_i = low$; in other words, if the ith level of the HDI factor is neither medium nor high, then it must be low.

It takes a little head-scratching to figure out what these indicators do, but **their effect in the model is to allow the intercept to change depending on the level of the factor**.

These indicators functions are sometimes called 'dummy' variables. They're simple to obtain using `pd.get_dummies(...)` :

In [36]:
```
# encode hdi factor by indicator variables
hdi_df = pd.get_dummies(reg_data.hdi_fac, drop_first = True)

# preview
hdi_df.head(6)
```

Out[36]:

Country	2_medium	3_high
Afghanistan	0	0
Albania	1	0
Algeria	1	0
Angola	0	0
Antigua and Barbuda	1	0
Argentina	0	1

Now compare this with the underlying factor:

In [37]:
```
reg_data[['hdi_fac']].head(6)
```

Out[37]:

Country	hdi_fac
Afghanistan	1_low
Albania	2_medium
Algeria	2_medium
Angola	1_low
Antigua and Barbuda	2_medium
Argentina	3_high

Interaction terms

We'll also add *interaction* terms -- a fancy phrase for products of explanatory variables -- between the indicators and the education variable. **The effect of interactions is to allow the slope to change depending on the level of the factor**.

The cell below computes these interaction terms and appends them, along with the indicators, to the education variable.

In [38]:
```
# compute interaction terms
interaction_df = hdi_df.multiply(reg_data.educ_expected_yrs_f, axis = 0)
interaction_df.columns = ['2_medium_x_educ', '3_high_x_educ']

# append indicators to data
x_mlr_df = pd.concat([reg_data.educ_expected_yrs_f, hdi_df, interaction_df], ignore_index = False, axis = 1)

# preview
x_mlr_df.head(4)
```

Out[38]:

Country	educ_expected_yrs_f	2_medium	3_high	2_medium_x_educ	3_high_x_educ
Afghanistan	6.795722	0	0	0.000000	0.0
Albania	13.201755	1	0	13.201755	0.0
Algeria	12.108990	1	0	12.108990	0.0
Angola	6.973901	0	0	0.000000	0.0

Notice that those interaction terms are simply the elementwise products of each indicator column with the education variable column.

Model fitting

With the encoded categorical variable for HDI, and the interaction terms, the linear model is:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 & x_1 & I(hdi_1 = med) & I(hdi_1 = high) & x_1 I(hdi_1 = med) & x_1 I(hdi_1 = high) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & I(hdi_n = med) & I(hdi_n = high) & x_n I(hdi_n = med) & x_n I(hdi_n = high) \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \end{bmatrix}}_{\beta} + \underbrace{\begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}}_{\epsilon}$$

Which is matrix form for:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 I(hdi_1 = med) + \beta_3 I(hdi_1 = high) + \beta_4 x_1 I(hdi_1 = med) + \beta_5 x_1 I(hdi_1 = high) + \epsilon_i \qquad i = 1, \dots, n$$

While seemingly more complicated, and perhaps a little tricker to interpret mathematically, the syntactical pattern for fitting the model is exactly the same as the simple linear model.

In [39]:
```
# add intercept column
x_mlr = add_dummy_feature(x_mlr_df, value = 1)

# configure module
mlr = LinearRegression(fit_intercept = False)

# fit model
mlr.fit(x_mlr, y)
```

Out[39]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

Q2 (a) Summarize model fit

Follow the examples from the simple linear model to carry out the following:

1. Compute the error variance estimate $\hat{\sigma}^2$ and store it in `sigma2_hat` .
2. Compute the standard errors for each coefficient and store in `mlrcoef_se` .
3. Construct a dataframe called `mlrcoef_table` showing both the estimates and standard errors.

(*Suggestion*: copy the codes from part 1 into a single cell below for reference.)

In [40]:
```
# store dimensions of explanatory variable matrix
n, p = x_mlr.shape

# compute residual variance
sigma2_hat = (y - mlr.predict(x_mlr)).var()*(n - 1)/(n - p)

# compute standard errors
mlrcoef_vcov = np.linalg.inv(x_mlr.transpose().dot(x_mlr)) * sigma2_hat
mlrcoef_se = np.sqrt(mlrcoef_vcov.diagonal())

# construct coefficient table
mlrcoef_table = pd.DataFrame(
    data = {'coefficient estimate': mlr.coef_, 'standard error': mlrcoef_se},
    index = ['intercept', 'education', 'medium', 'high', 'edu_times_med', 'edu_times_high']
)

# print
mlrcoef_table
```

Out[40]:

	coefficient estimate	standard error
intercept	7.049013	0.369182
education	-0.351265	0.041062
medium	-2.057722	0.898528
high	-4.504779	2.485328
edu_times_med	0.125183	0.079162
edu_times_high	0.286587	0.187939

In [41]:
```
grader.check("q2_a")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-41-3bd349818c3a> in <module>
----> 1 grader.check("q2_a")

/opt/conda/lib/python3.7/site-packages/otter/check/utils.py in run_function(self, *args, **kwargs)
    129                 except Exception as e:
    130                     self._log_event(event_type, success=False, error=e)
--> 131                     raise e
    132                 else:
    133                     self._log_event(event_type, results=results, question=question, shelve_env=shelve_env)

/opt/conda/lib/python3.7/site-packages/otter/check/utils.py in run_function(self, *args, **kwargs)
    122                 try:
    123                     if event_type == EventType.CHECK:
--> 124                         question, results, shelve_env = f(self, *args, **kwargs)
    125                     else:
    126                         results = f(self, *args, **kwargs)

/opt/conda/lib/python3.7/site-packages/otter/check/notebook.py in check(self, question, global_env)
    177 
    178         # run the check
--> 179         result = check(test_path, test_name, global_env)
    180 
    181         return question, result, global_env

/opt/conda/lib/python3.7/site-packages/otter/execute/__init__.py in check(nb_or_test_path, test_name, global_env)
     44         test = OKTestFile.from_file(nb_or_test_path)
     45     else:
---> 46         test = NotebookMetadataOKTestFile.from_file(nb_or_test_path, test_name)
     47 
     48     if global_env is None:

/opt/conda/lib/python3.7/site-packages/otter/test_files/metadata_test.py in from_file(cls, path, test_name)
     74             nb = json.load(f)
     75 
---> 76         test_spec = nb["metadata"][NOTEBOOK_METADATA_KEY]["tests"]
     77         if test_name not in test_spec:
     78             raise ValueError(f"Test {test_name} not found")

KeyError: 'otter'
```

Q2 (b) Fit metric

Now let's compare the fit of the MLR model to that of the SLR model.

(i) Calculate the R^2 score for the multiple linear regression model and store it in `R_2`.

In [42]:
```
# solution
R_2 = r2_score(y, mlr.predict(x_mlr))

R_2
```

Out[42]: 0.7657210526482656

In [43]:
```
grader.check("q2_b_i")
```

Out[43]: **q2_b_i** passed!

(ii) Does the fit seem to have improved appreciably?

Answer

Yes, the R^2 score has improved about 0.04.

Q2 (c) Visualize

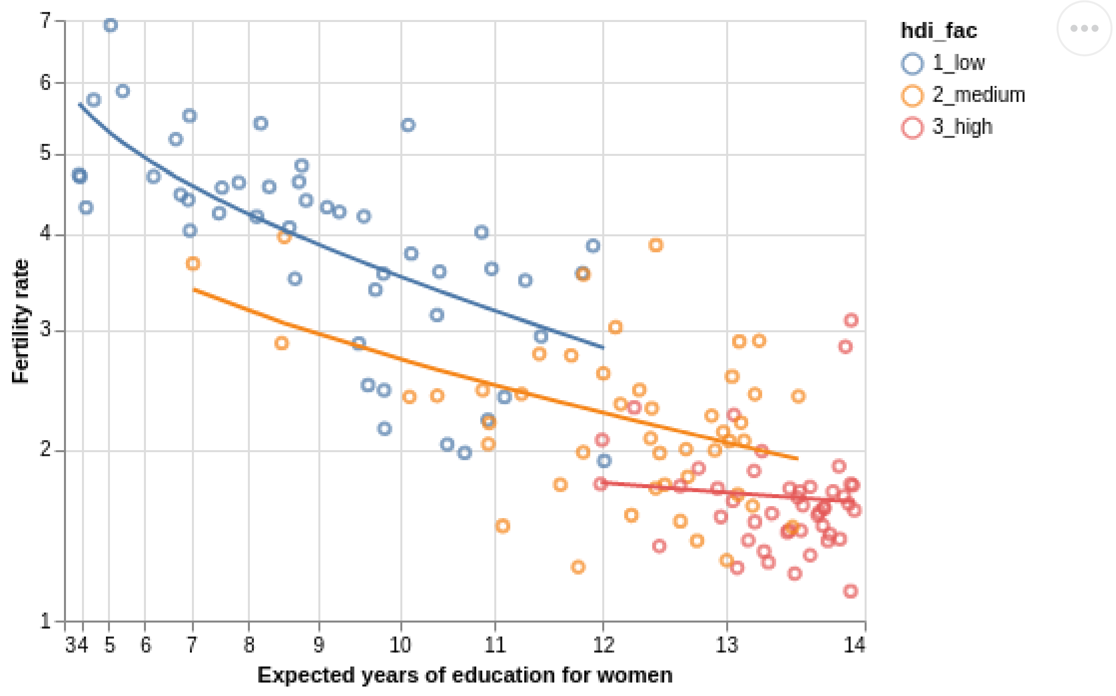
Now visualize the model outputs.

1. First append the fitted values (if you didn't above) as a new column named `fitted_mlr` to `reg_data`.
2. Construct a line plot of the fitted values (y axis) against education (x axis) and map the HDI factor to the color aesthetic. This should produce a plot with three lines, one for each level of HDI. Store the result as `mlr_lines`.
3. Layer `mlr_lines` on top of `scatter_scaled`.

In [44]:
```
# append fitted values to original data
reg_data['fitted_mlr'] = mlr.predict(x_mlr)

# construct line plot
mlr_lines = alt.Chart(reg_data).mark_line().encode(
    x = alt.X('educ_expected_yrs_f', scale = alt.Scale(zero = False)),
    y = alt.Y('fitted_mlr', scale = alt.Scale(zero = False)),
    color = alt.Color('hdi_fac')
)
# layer
scatter_scaled + mlr_lines
```

Out[44]:



Q2 (d) Interpret your graphic

What does the multiple linear model suggest about the relationship between fertility rate and expected years of education for high-HDI countries as compared with low-HDI countries?

(*Hint*: focus on how the slopes of the two lines compare.)

Answer

As the expected years of education for women grows, the fertility rate for low-HDI countries tend to reduce faster than high-HDI countries. From the plot we can see that the slope for low-HDI countries is much smaller than that of high-HDI countries.

Comment

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.

Although we can't claim that, for example, 'the mean fertility rate decreases with education at a rate of -0.35 children per woman per expected year of education in low-HDI countries', we *can* say '*among the countries reporting data*, the mean fertility rate decreases with education at a rate of -0.35 children per woman per expected year of education in low-HDI countries'. This is a nice example of how a model might be used in a descriptive capacity.

Submission Checklist

1. Save file to confirm all changes are on disk
2. Run *Kernel* > *Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File* > *Download as* > *HTML*.
5. Open in Google Chrome and print to PDF on A3 paper in portrait orientation.
6. Submit to Gradescope

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

In [45]:
```
grader.check_all()
```
Out[45]:
```
q1_a_i results: All test cases passed!

q1_a_ii results: All test cases passed!

q1_b results: All test cases passed!

q1_c results: All test cases passed!

q1_d results: All test cases passed!

q2_b_i results: All test cases passed!
```