In [1]:
```
from datascience import *
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
```

In [2]:
```
def draw_line(slope=0, intercept=0, x=None, color='r'):
    if x is None:
        x1, x2, y1, y2 = plots.gca().axis()
    x = make_array(x1, x2)
    y = x*slope + intercept
    plots.plot(x, y, color=color)
```

In [3]:
```
def demographics_errors(slope, intercept):
    sample = [[14.7, 33995], [19.1, 61454], [50.7, 71183], [59.5, 105918]]
    demographics.scatter('College%', 'Median Income', alpha=0.5)
    xlims = make_array(5, 75)
    plots.plot(xlims, slope * xlims + intercept, lw=4)
    for x, y in sample:
        plots.plot([x, x], [y, slope * x + intercept], color='r', lw=4)
```

Regression Line

In [4]:
```
def standard_units(arr):
    return (arr - np.average(arr))/np.std(arr)

def correlation(t, x, y):
    x_standard = standard_units(t.column(x))
    y_standard = standard_units(t.column(y))
    return np.average(x_standard * y_standard)

def slope(t, x, y):
    r = correlation(t, x, y)
    y_sd = np.std(t.column(y))
    x_sd = np.std(t.column(x))
    return r * y_sd / x_sd

def intercept(t, x, y):
    x_mean = np.mean(t.column(x))
```

```
    y_mean = np.mean(t.column(y))
    return y_mean - slope(t, x, y)*x_mean
```

In [5]:
```
def fitted_values(t, x, y):
    """Return an array of the regressions estimates at all the x values"""
    a = slope(t, x, y)
    b = intercept(t, x, y)
    return a*t.column(x) + b
```

In [6]:
```
demographics = Table.read_table('district_demographics2016.csv').drop(3)
demographics.show(5)
```

State	District	Median Income	College%
Alabama	Congressional District 1 (115th Congress), Alabama	47083	24
Alabama	Congressional District 2 (115th Congress), Alabama	42035	21.8
Alabama	Congressional District 3 (115th Congress), Alabama	46544	22.8
Alabama	Congressional District 4 (115th Congress), Alabama	41110	17
Alabama	Congressional District 5 (115th Congress), Alabama	51690	30.3

... (430 rows omitted)

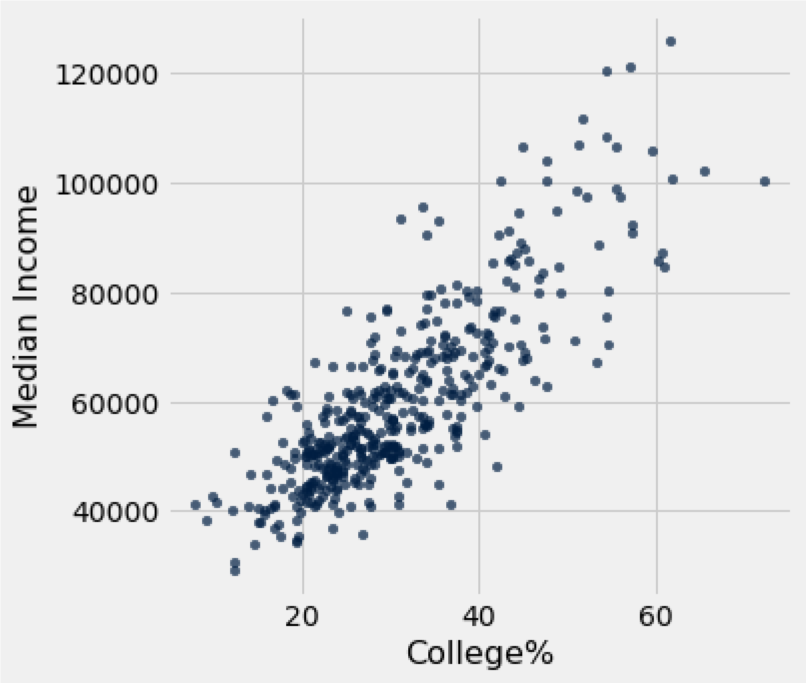
In [7]:
```
demographics = demographics.drop(
    'State', 'District')
demographics.show(5)
```

Median Income	College%
47083	24
42035	21.8
46544	22.8
41110	17
51690	30.3

... (430 rows omitted)

In [8]:

```
demographics.scatter('College%', 'Median Income')
```



In [9]:

```
correlation(demographics, 'College%', 'Median Income')
```

Out[9]:

```
0.8184648517141335
```

In [10]:

```
regression_slope = slope(demographics, 'College%', 'Median Income')
regression_intercept = intercept(demographics, 'College%', 'Median Income')
(regression_slope, regression_intercept)
```

Out[10]:

```
(1270.70168946388, 20802.577766677925)
```

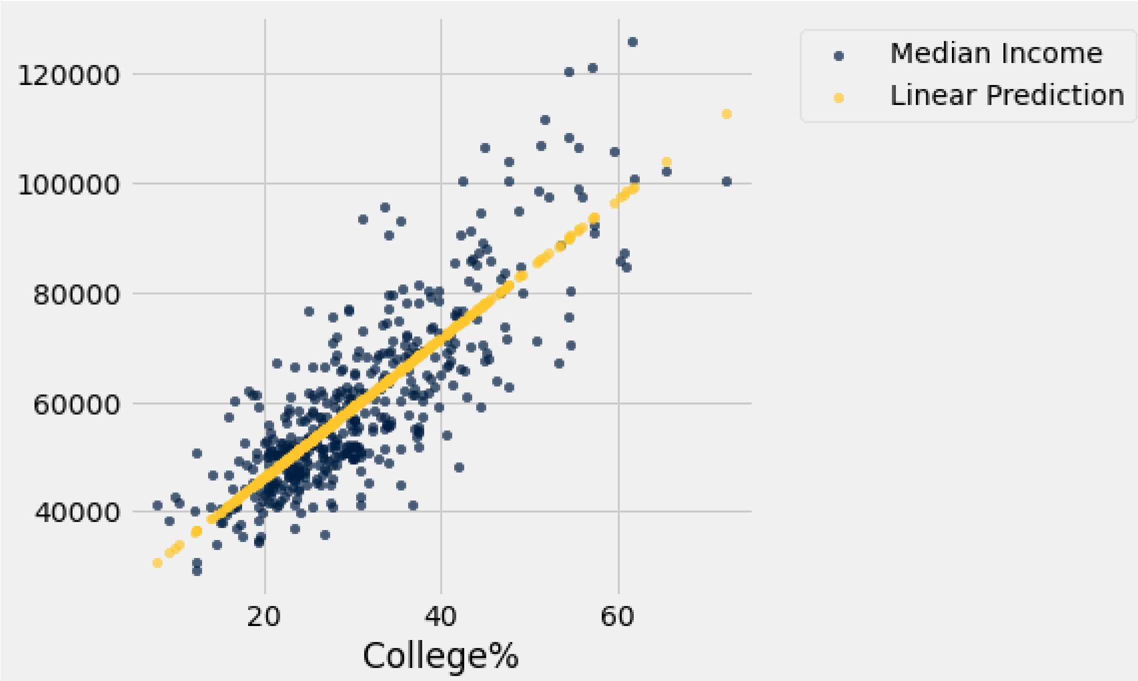
In [11]:

```
predicted = fitted_values(demographics, 'College%', 'Median Income')
```

In [12]:

```
demographics = demographics.with_column(
```

```
    'Linear Prediction', predicted)
demographics.scatter('College%')
```



In [13]:

```
actual = demographics.column('Median Income')
errors = actual - predicted
```

In [14]:

```
demographics.with_column('Error', errors)
```

Out[14]:

Median Income	College%	Linear Prediction	Error
47083	24	51299.4	-4216.42
42035	21.8	48503.9	-6468.87
46544	22.8	49774.6	-3230.58
41110	17	42404.5	-1294.51
51690	30.3	59304.8	-7614.84
61413	36.7	67437.3	-6024.33

Median Income	College%	Linear Prediction	Error
34664	19.4	45454.2	-10790.2
76440	29.6	58415.3	18024.7
50537	24.5	51934.8	-1397.77
49072	34	64006.4	-14934.4

... (425 rows omitted)

In [15]:

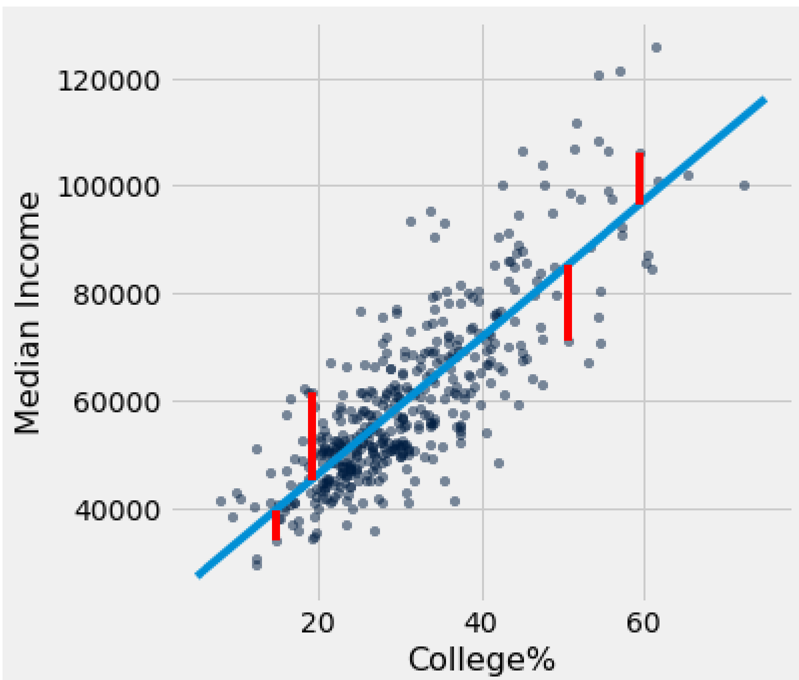
```
np.mean(errors ** 2) ** 0.5
```

Out[15]:

```
9398.515588571281
```

In [16]:

```
demographics_errors(regression_slope, regression_intercept)
```

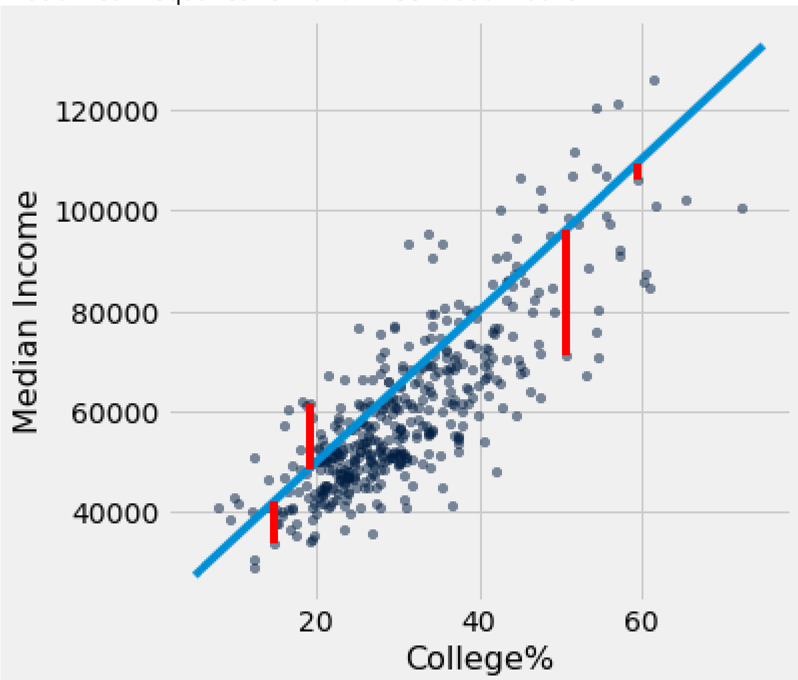


Root Mean Square Error

In [17]:
```
def show_demographics_rmse(slope, intercept):
    demographics_errors(slope, intercept)
    x = demographics.column('College%')
    y = demographics.column('Median Income')
    prediction = slope * x + intercept
    mse = np.mean((y - prediction) ** 2)
    print("Root mean squared error:", mse ** 0.5)
```

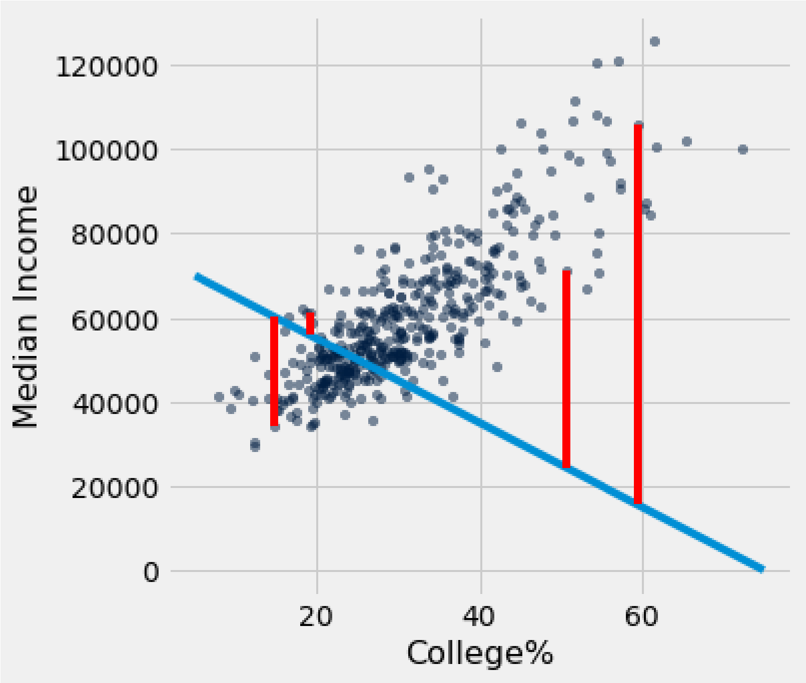
In [18]:
```
show_demographics_rmse(1500, 20000)
```

```
Root mean squared error: 11559.086490075999
```



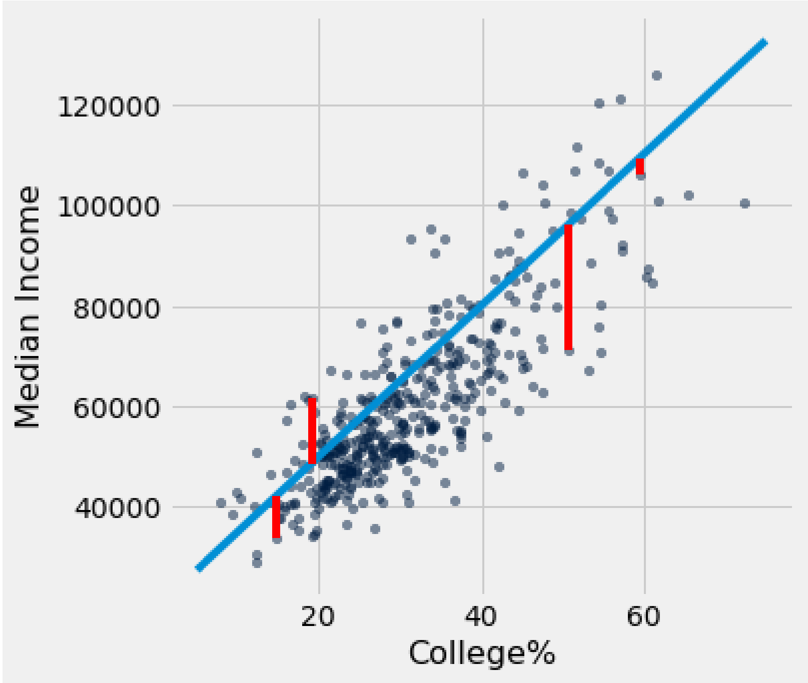
In [19]:
```
show_demographics_rmse(-1000, 75000)
```

```
Root mean squared error: 30247.883767944502
```



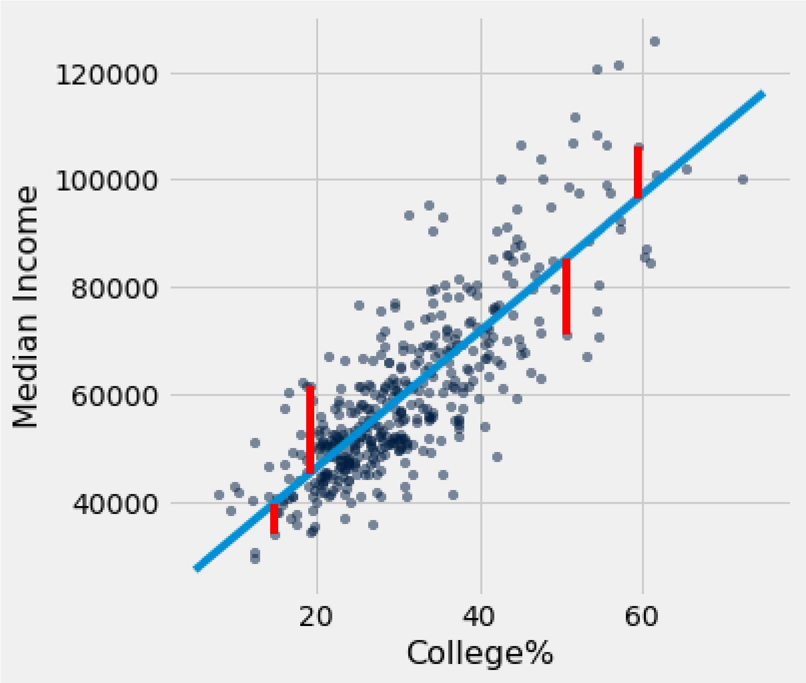
In [20]:
```
show_demographics_rmse(1500, 20000)
```

```
Root mean squared error: 11559.086490075999
```



In [21]:
```
show_demographics_rmse(regression_slope, regression_intercept)
```

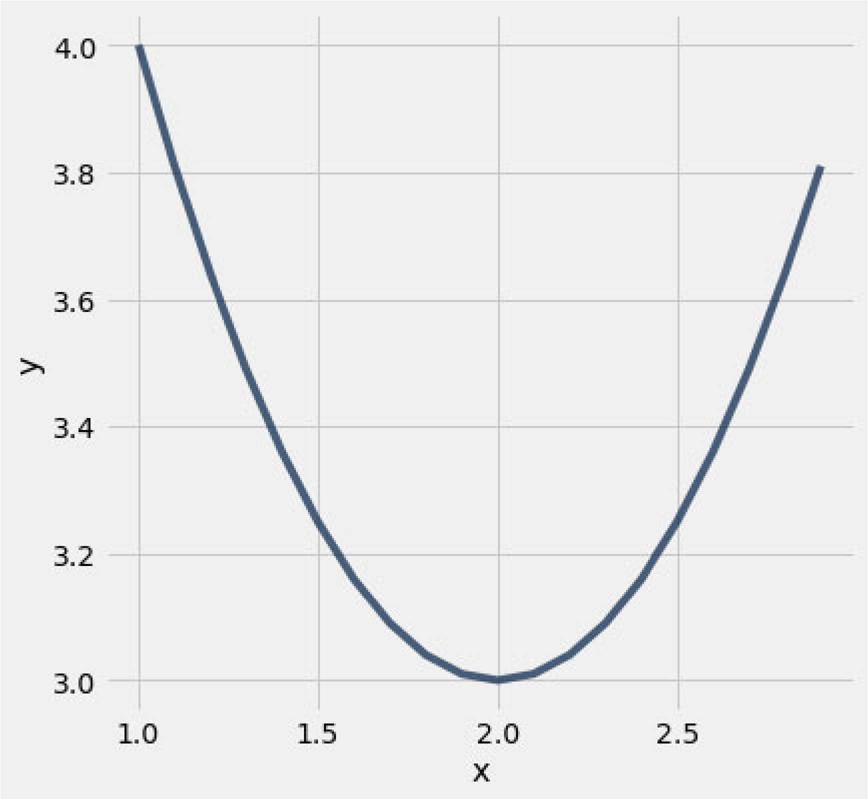
```
Root mean squared error: 9398.515588571281
```



Numerical Optimization

In [22]:

```
x = np.arange(1, 3, 0.1)
y = (x-2)**2 + 3
Table().with_columns('x', x, 'y', y).plot('x')
```



In [23]:
```
def f(x):
    return ((x-2)**2) + 3
```

In [24]:
```
minimize(f)
```

Out[24]: 1.9999999946252267

Minimizing RMSE

In [25]:
```
def demographics_rmse(any_slope, any_intercept):
    x = demographics.column('College%')
    y = demographics.column('Median Income')
    estimate = any_slope*x + any_intercept
    return (np.mean((y - estimate) ** 2)) ** 0.5
```

In [26]:
```
minimize(demographics_rmse)
```

Out[26]:
```
array([ 1270.70168805, 20802.57933807])
```

In [27]:
```
make_array(regression_slope, regression_intercept)
```

Out[27]:
```
array([ 1270.70168946, 20802.57776668])
```

Nonlinear Regression

In [28]:
```
shotput = Table.read_table('shotput.csv')
```

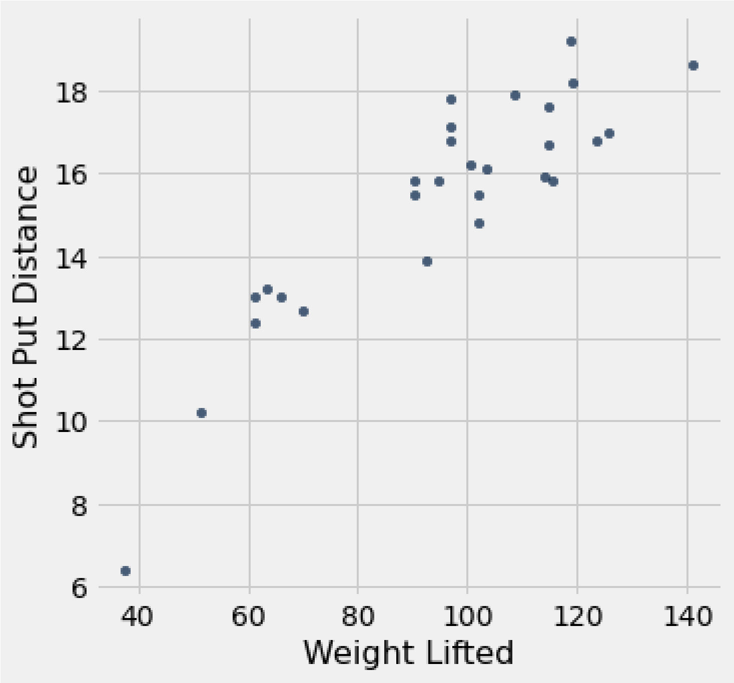
In [29]:
```
shotput
```

Out[29]:

Weight Lifted	Shot Put Distance
37.5	6.4
51.5	10.2
61.3	12.4
61.3	13
63.6	13.2
66.1	13
70	12.7
92.7	13.9
90.5	15.5
90.5	15.8

... (18 rows omitted)

In [30]:
```
shotput.scatter('Weight Lifted')
```



In [31]:
```
def shotput_linear_rmse(any_slope, any_intercept):
    x = shotput.column('Weight Lifted')
    y = shotput.column('Shot Put Distance')
    estimate = any_slope*x + any_intercept
    return np.mean((y - estimate) ** 2) ** 0.5
```

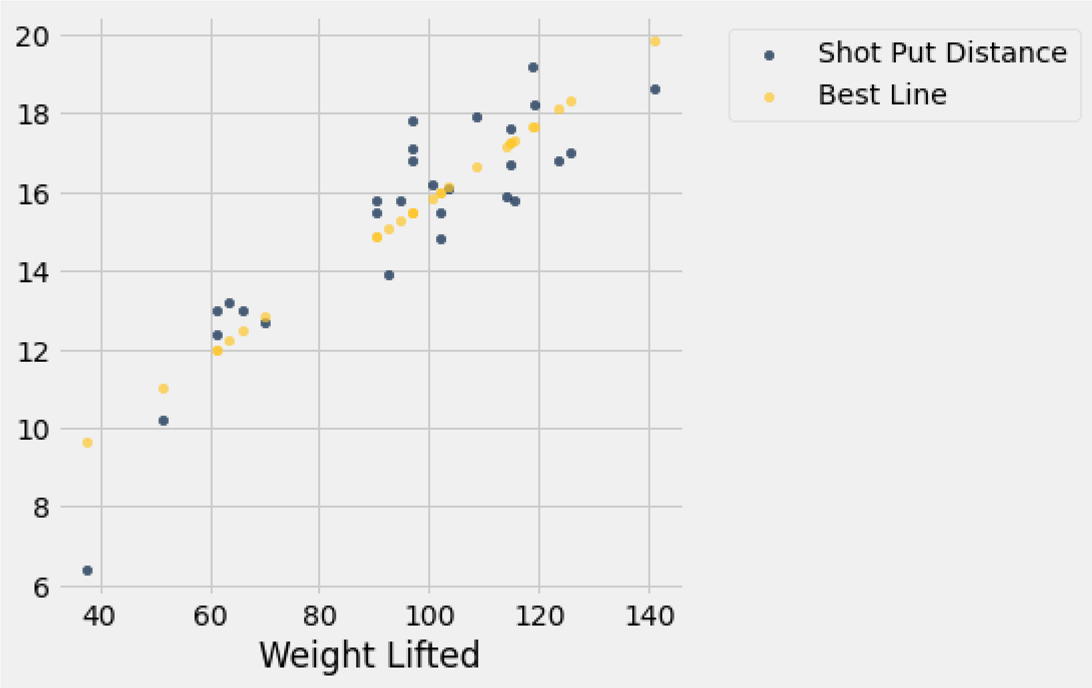
In [32]:
```
best_line = minimize(shotput_linear_rmse)
best_line
```

Out[32]:
```
array([0.09834382, 5.95962883])
```

In [33]:
```
weights = shotput.column(0)
```

In [34]:
```
linear_fit = best_line.item(0)*weights + best_line.item(1)

shotput.with_column(
    'Best Line', linear_fit
).scatter(0)
```



Quadratic Function

$$f(x) \ = \ ax^2 + bx + c$$

for constants a, b, and c.

In [35]:

```
def shotput_quadratic_rmse(a, b, c):
    x = shotput.column('Weight Lifted')
    y = shotput.column('Shot Put Distance')
    estimate = a*(x**2) + b*x + c
    return np.mean((y - estimate) ** 2) ** 0.5
```

In [36]:

```
best_quad = minimize(shotput_quadratic_rmse)
best_quad
```

Out[36]:

```
array([-1.04003731e-03,  2.82706003e-01, -1.53167618e+00])
```

In [37]:

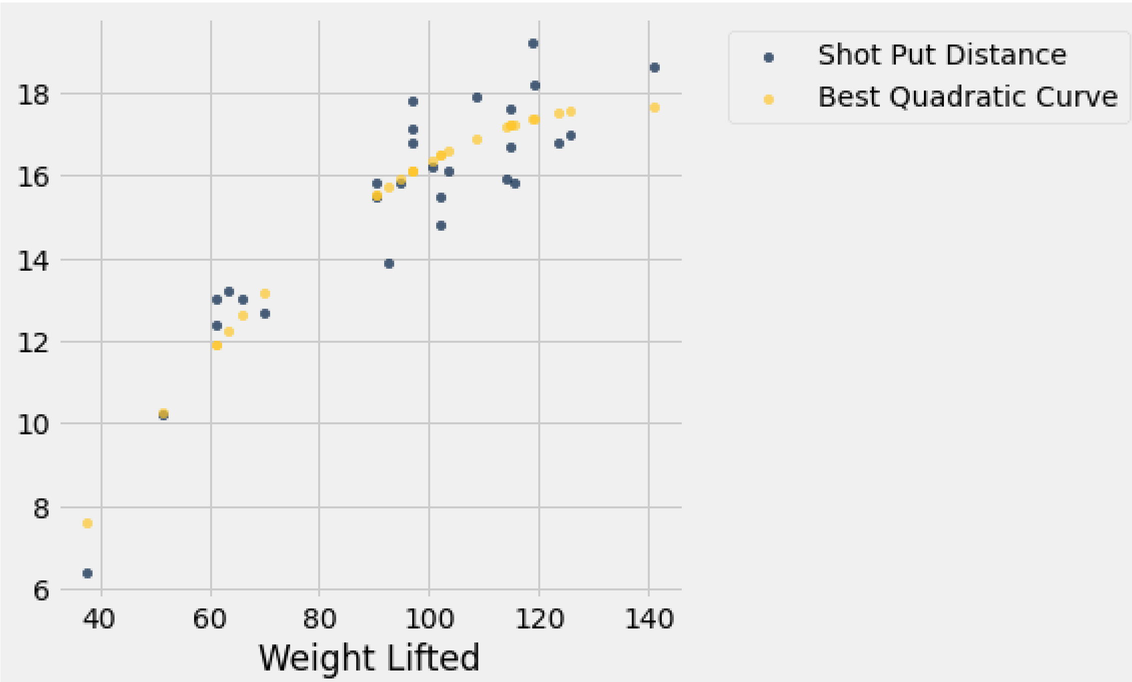
```
# x = weight lifted = 100 kg
# Then predicted shot put distance:
```

```
(-0.00104)*(100**2) + 0.2827*100 - 1.5318
```

Out[37]: 16.3382

In [38]:
```
quad_fit = best_quad.item(0)*(weights**2) + best_quad.item(1)*weights + best_quad.item(2)
```

In [39]:
```
shotput.with_column('Best Quadratic Curve', quad_fit).scatter(0)
```



In [ ]:

In [ ]: