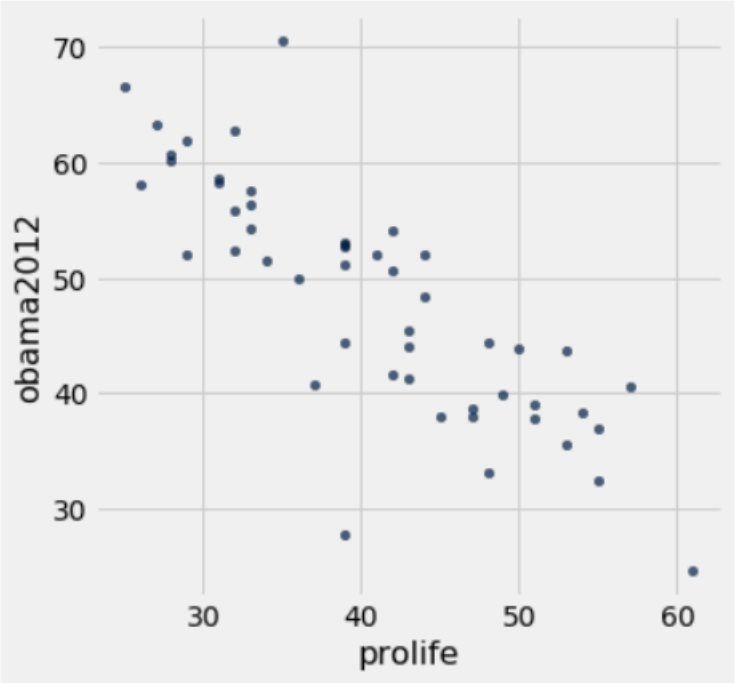
1. a)

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

```
In [2]: states_data = Table.read_table("states_data.csv")
```

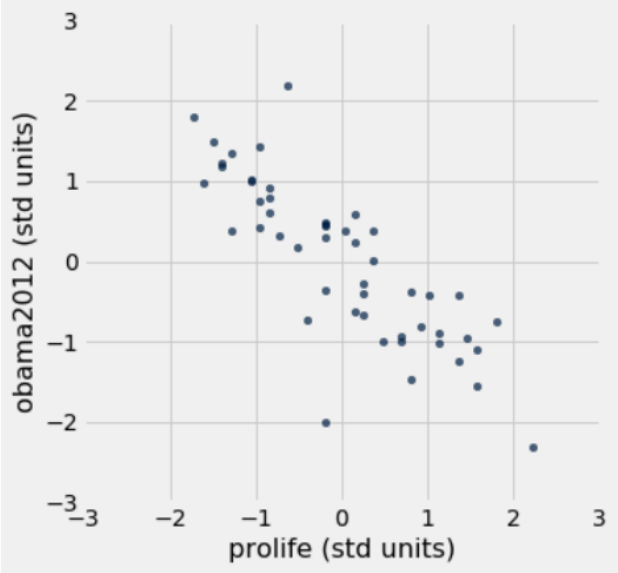
```
In [3]: states_data.scatter('prolife', 'obama2012')
```



There's a moderately strong negative relationship between the percentage of people in a state with pro-life views and the percentage of the presidential vote earned by Obama in 2012 in that state.

b)

```
In [4]: def standard_units(any_numbers):
            "Convert any array of numbers to standard units."
            return (any_numbers - np.mean(any_numbers))/np.std(any_numbers)


        Table().with_columns(
             'prolife (std units)', standard_units(states_data.column('prolife')),
            'obama2012 (std units)',  standard_units(states_data.column('obama2012')),

        ).scatter(0, 1)
        plots.xlim(-3, 3)
        plots.ylim(-3, 3);
```



No. Because there's no point at (0, 0) in the graph.

c)

```
In [5]: def correlation(t, x, y):
            return np.mean(standard_units(t.column(x))*standard_units(t.column(y)))

        print("\n correlation btwn obama2012 and prolife is", correlation(states_data, "obama2012", "prolife") )
```

```
correlation btwn obama2012 and prolife is -0.8283501573452499
```

d)

```
In [6]: print("\n correlation btwn prolife and obama2012 is", correlation(states_data, "prolife", "obama2012") )
```

```
correlation btwn prolife and obama2012 is -0.8283501573452499
```

The two are the same, so the correlation between variables is symmetric.

e)

```
In [8]: pro2 = (states_data.column("prolife")*45)-5
        obama20122 = (states_data.column("obama2012"))+3


        temp_table = Table().with_columns([
              'pro2', pro2,
            'obama20122',  obama20122, ])

        print("\n correlation btwn pro2 and obama20122 is", correlation(temp_table, "pro2", "obama20122") )
```

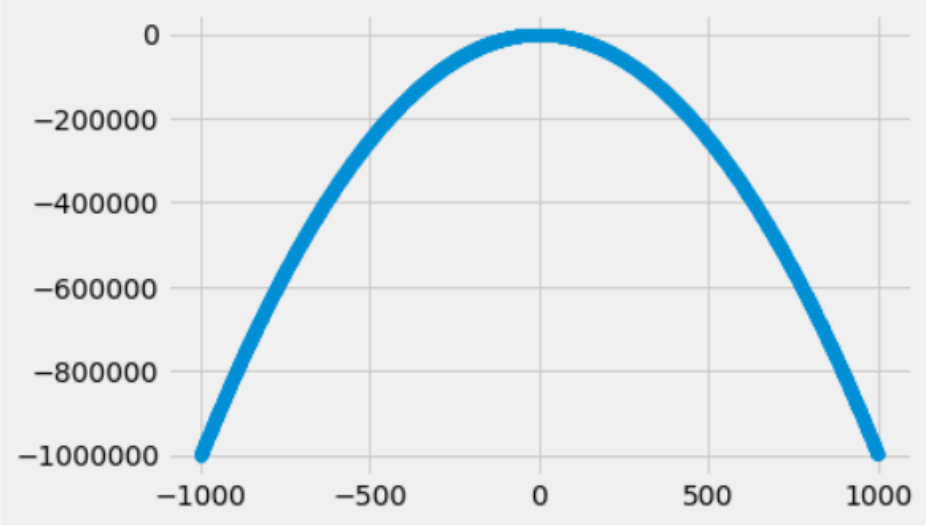
```
correlation btwn pro2 and obama20122 is -0.8283501573452499
```

I found that correlation doesn't change after rescaling. It's because linear rescaling does not affect the correlation between the variables. When calculating the correlation coefficient, the variables are standardized. So changes in scale does not matter.

f)

```
In [9]: x = np.arange(-1000, 1000)
        y = -x**2

        plots.scatter(x, y)

        print("\n correlation btwn x and y is", np.corrcoef(x, y)[1,0] )
```

```
correlation btwn x and y is 0.0019364890104331498
```



They are related, but it's not a linear relationship. The correlation is basically zero. Because correlation only measures linear association. Variables that have strong non-linear association might have very low correlation.

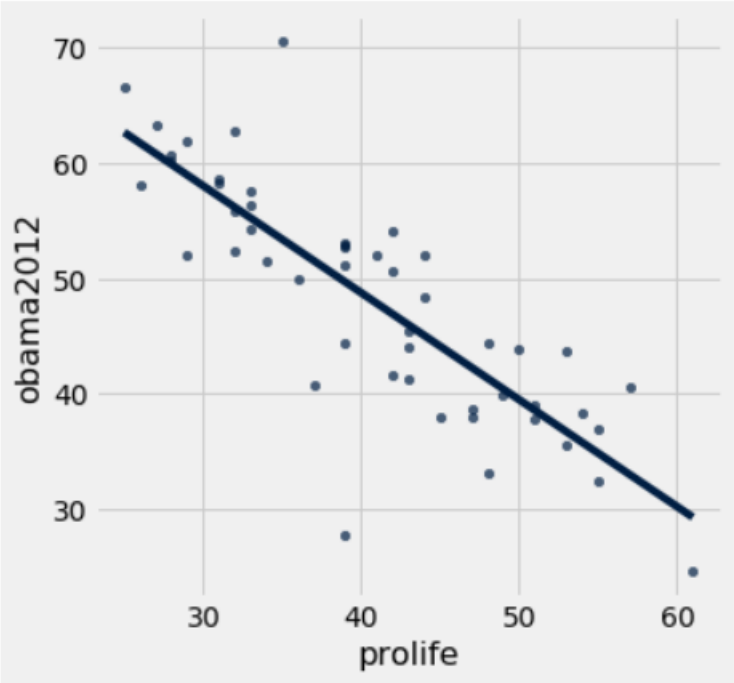
g)

```
In [10]: def slope(t, label_x, label_y):
             r = correlation(t, label_x, label_y)
             return r*np.std(t.column(label_y))/np.std(t.column(label_x))

         def intercept(t, label_x, label_y):
             return np.mean(t.column(label_y)) - slope(t, label_x, label_y)*np.mean(t.column(label_x))

         # run regression
         model_slope = slope(states_data, 'prolife', 'obama2012')
         model_intercept = intercept(states_data, 'prolife', 'obama2012')
         model_intercept, model_slope

Out[10]: (85.80142266610778, -0.9258687325653661)
```

When the percentage of people in a state with pro-life views is zero, the percentage of the expected presidential vote earned by Obama in 2012 in that state would be basically 85.8014%. When the percentage of people in a state with pro-life views increases by one percent, the expected percentage of the presidential vote earned by Obama in 2012 in that state would decrease by basically 0.9259%.

h)

```
In [11]: def fit(table, x, y):
             a = slope(table, x, y)
             b = intercept(table, x, y)
             return a * table.column(x) + b

         states_data.scatter('prolife', 'obama2012', fit_line=True)
```



i)

In [13]:
```
states_with_predictions = states_data.with_column(
    'Regression Prediction', model_slope*states_data.column('prolife') + model_intercept
)

states_with_predictions["stateid"]
NJS = states_with_predictions.where(states_with_predictions["stateid"] == 'NJ    ')
print("New Jersey predicted obama2012:", NJS["Regression Prediction"],"")

NJ_actual = states_data.where(states_data["stateid"]=='NJ    ')["obama2012"]
print("New Jersey actual obama2012:", NJ_actual,"")
```

```
New Jersey predicted obama2012: [57.09949196]
New Jersey actual obama2012: [58.25]
```

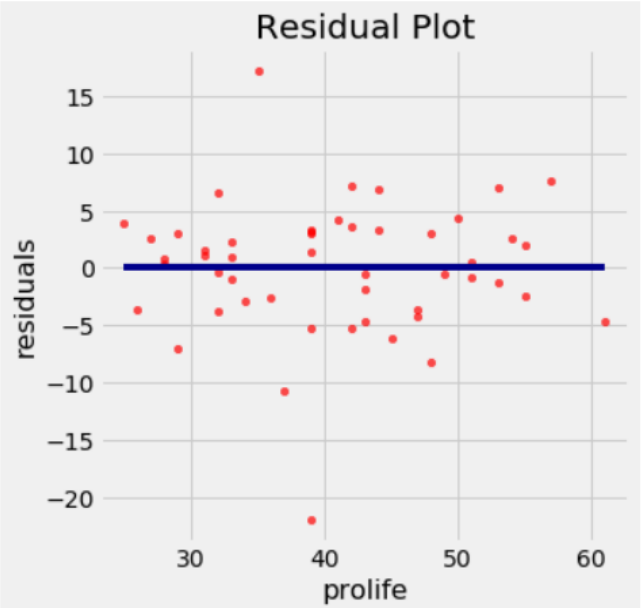
j)

In [14]:
```
def residual(table, x, y):
    return table.column(y) - fit(table, x, y)

CT_resid = residual(states_data, "prolife", "obama2012")[states_data["stateid"]=='CT    ']
print("Connecticut regression residual:", CT_resid,"")
```

```
Connecticut regression residual: [-3.66883562]
```

k)

In [15]:
```
def residual_plot(table, x, y):
    x_array = table.column(x)
    t = Table().with_columns(
            x, x_array,
            'residuals', residual(table, x, y)
        )
    t.scatter(x, 'residuals', color='r')
    xlims = make_array(min(x_array), max(x_array))
    plots.plot(xlims, make_array(0, 0), color='darkblue', lw=4)
    plots.title('Residual Plot')

residual_plot(states_data, 'prolife', 'obama2012')
```



For range between 50% to 60% people in a state with pro-life views, the residual appears to be a cone-like shape. The residual is getting larger. The variability of “prolife” is unequal across the range.

l)

```
In [16]: def r2(t, x, y):
             return correlation(t, x, y)**2

         r2(states_data, "prolife", "obama2012")
```

```
Out[16]: 0.6861639831739001
```

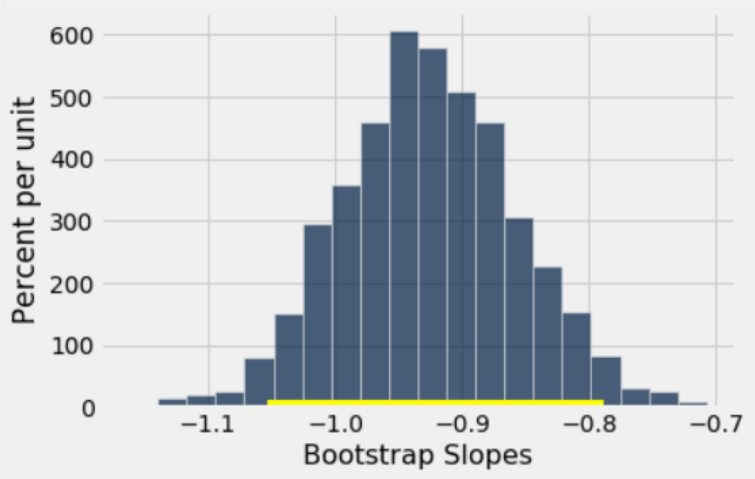
R^2 equals 0.6861639831739001 means the model explains basically 68.6164% of the variability of the response data around its mean.

m)

```
In [17]: def bootstrap_slope(table, x, y, repetitions):

             slopes = make_array()
             for i in np.arange(repetitions):
                 bootstrap_sample = table.sample()
                 bootstrap_slope = slope(bootstrap_sample, x, y)
                 slopes = np.append(slopes, bootstrap_slope)

             left = percentile(2.5, slopes)
             right = percentile(97.5, slopes)

             observed_slope = slope(table, x, y)

             Table().with_column('Bootstrap Slopes', slopes).hist(bins=20)
             plots.plot(make_array(left, right), make_array(0, 0), color='yellow', lw=8);
             print('Slope of regression line:', observed_slope)
             print('Approximate 95%-confidence interval for the true slope:')
             print(left, right)
```

```
In [18]: bootstrap_slope(states_data, 'prolife', 'obama2012', 2000)
```

```
Slope of regression line: -0.9258687325653661
Approximate 95%-confidence interval for the true slope:
-1.0532762727360214 -0.7886072161697801
```



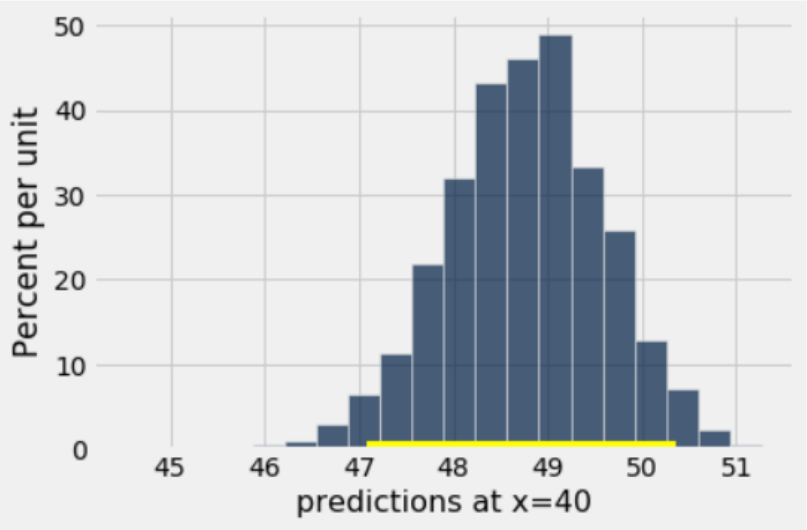
Yes, it is statistically significant. Because the 95% confidence interval for the true slope [-1.0532762727360214 -0.7886072161697801] includes the true slope of the regression line, -0.9258687325653661.

n)

In [17]:
```
def fitted_value(table, x, y, given_x):
    a = slope(table, x, y)
    b = intercept(table, x, y)
    return a * given_x  + b

def bootstrap_prediction(table, x, y, new_x, repetitions):

    predictions = make_array()
    for i in np.arange(repetitions):
        bootstrap_sample = table.sample()
        bootstrap_prediction = fitted_value(bootstrap_sample, x, y, new_x)
        predictions = np.append(predictions, bootstrap_prediction)

    left = percentile(2.5, predictions)
    right = percentile(97.5, predictions)

    original = fitted_value(table, x, y, new_x)

    Table().with_column('Prediction', predictions).hist(bins=20)
    plots.xlabel('predictions at x='+str(new_x))
    plots.plot(make_array(left, right), make_array(0, 0), color='yellow', lw=8);
    print('Height of regression line at x='+str(new_x)+':', original)
    print('Approximate 95%-confidence interval:')
    print(left, right)
```

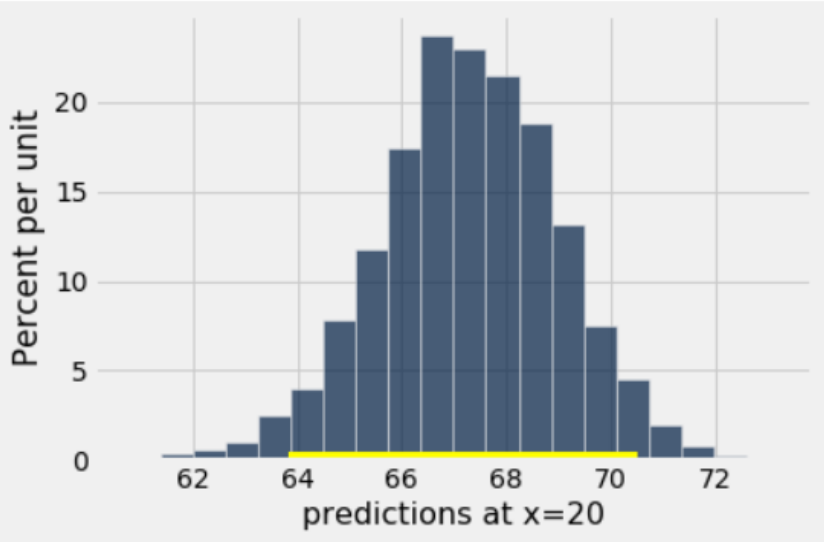
In [20]:
```
bootstrap_prediction(states_data, 'prolife', 'obama2012', 40, 3000)
```

```
Height of regression line at x=40: 48.76667336349314
Approximate 95%-confidence interval:
47.08093073593075 50.35083791958688
```



In [21]:
```
bootstrap_prediction(states_data, 'prolife', 'obama2012', 20, 3000)
```

```
Height of regression line at x=20: 67.28404801480046
Approximate 95%-confidence interval:
63.83867553616939 70.5080748633244
```



The one with x=20 is wider. Because with a larger sample size x=40, we can make a more specific prediction for confidence interval.

2. a)

In [22]:
```
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

def all_distances(training, point):
    attributes = training.drop('Class')
    def distance_from_point(row):
        return distance(point, np.array(row))
    return attributes.apply(distance_from_point)

def table_with_distances(training, point):
    return training.with_column('Distance', all_distances(training, point))

def closest(training, point, k):
    with_dists = table_with_distances(training, point)
    sorted_by_distance = with_dists.sort('Distance')
    topk = sorted_by_distance.take(np.arange(k))
    return topk

def majority(topkclasses):
    ones = topkclasses.where('Class', are.equal_to(1)).num_rows
    zeros = topkclasses.where('Class', are.equal_to(0)).num_rows
    if ones > zeros:
        return 1
    else:
        return 0

def classify(training, p, k):
    closestk = closest(training, p, k)
    topkclasses = closestk.select('Class')
    return majority(topkclasses)
```

In [23]:
```
ckd = Table.read_table('ckd.csv')
```

In [25]:
```
ckd_small = Table().with_columns(
    'Age', standard_units(ckd.column('Age')),
    'Sodium', standard_units(ckd.column('Sodium')),
    'Class', ckd.column('Class')
)
```

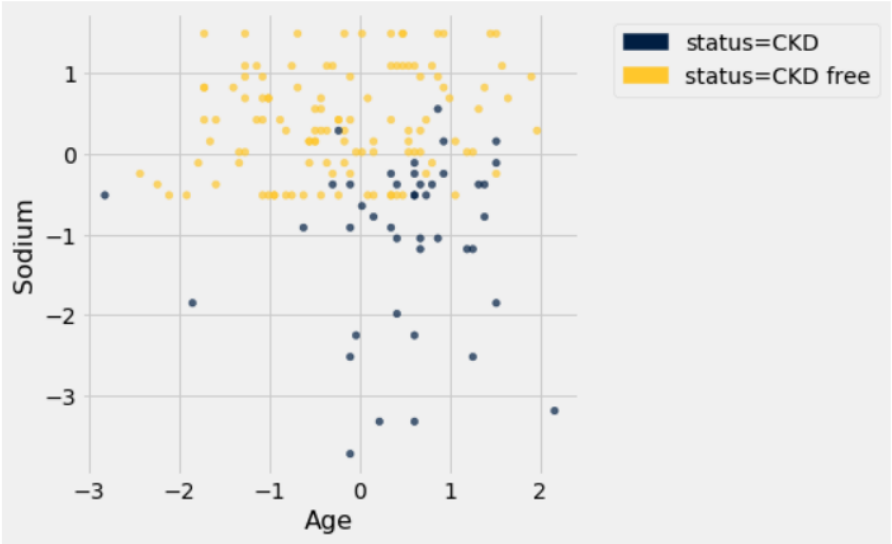
In [26]:
```
color_table = Table().with_columns(
    'Class', make_array(1, 0),
    'Color', make_array('blue', 'gold')
)
ckd_small = ckd_small.join('Class', color_table)


status_table = Table().with_columns(
    'Class', make_array(1, 0),
    'status', make_array("CKD","CKD free")
)
ckd_small = ckd_small.join('Class', status_table)

ckd_small
```

Out[26]:

Class	Age	Sodium	Color	status
0	-0.61846	-0.515439	gold	CKD free
0	-1.71785	1.49375	gold	CKD free
0	-0.295109	1.09192	gold	CKD free
0	0.480933	-0.515439	gold	CKD free
0	0.0929122	-0.515439	gold	CKD free
0	-1.00648	0.690077	gold	CKD free
0	0.674944	0.95797	gold	CKD free
0	-0.7478	-0.515439	gold	CKD free
0	-0.489119	0.154292	gold	CKD free
0	-0.941811	-0.515439	gold	CKD free

... (148 rows omitted)

In [27]:
```
ckd_small.scatter('Age', 'Sodium', group='status')
```



Generally speaking, sodium and age are both key factors of CKD. Low Sodium and high age is a bad combination.

b)

In [28]:
```
x_array = make_array()
y_array = make_array()
for x in np.arange(-2, 2.1, 0.1):
    for y in np.arange(-2, 2.1, 0.1):
        x_array = np.append(x_array, x)
        y_array = np.append(y_array, y)

test_grid = Table().with_columns(
    'Age', x_array,
    'Sodium', y_array
)
```

In [29]:
```
def classify_grid(training, test, k):
    c = make_array()
    for i in range(test.num_rows):
        c = np.append(c, classify(training, make_array(test.row(i)), k))
    return c

c = classify_grid(ckd_small.drop('status','Color'), test_grid, 1)
```

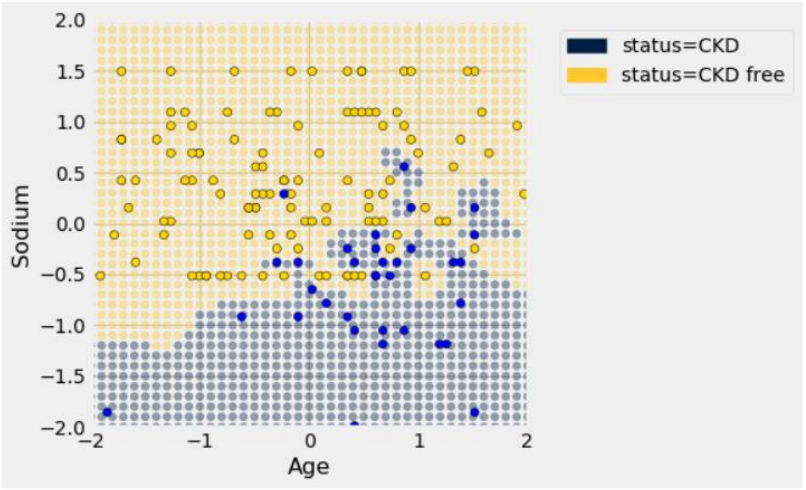
In [30]:
```
test_grid = test_grid.with_column('Class', c).join('Class', status_table)

test_grid
```

Out[30]:

Class	Age	Sodium	status
0	-2	-1.1	CKD free
0	-2	-1	CKD free
0	-2	-0.9	CKD free
0	-2	-0.8	CKD free
0	-2	-0.7	CKD free
0	-2	-0.6	CKD free
0	-2	-0.5	CKD free
0	-2	-0.4	CKD free
0	-2	-0.3	CKD free
0	-2	-0.2	CKD free

... (1671 rows omitted)

In [31]:
```
test_grid.scatter('Age', 'Sodium', group='status', alpha=0.4, s=30)

plots.scatter(ckd_small.column('Age'), ckd_small.column('Sodium'), c=ckd_small.column('Color'), edgecolor='k')
plots.xlim(-2, 2)
plots.ylim(-2, 2);
```



c)

In [32]:
```
new_patient = make_array(0.8, -0.25 )

ckd_attributes = ckd_small.select('Age', 'Sodium')

ckd_attributes
```

Out[32]:

Age	Sodium
-0.61846	-0.515439
-1.71785	1.49375
-0.295109	1.09192
0.480933	-0.515439
0.0929122	-0.515439
-1.00648	0.690077
0.674944	0.95797
-0.7478	-0.515439
-0.489119	0.154292
-0.941811	-0.515439

... (148 rows omitted)

In [33]:
```
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

def distance_from_new_patient(row):
    return distance(new_patient, np.array(row))

distances = ckd_attributes.apply(distance_from_new_patient)

ckd_with_distances = ckd_small.with_column('Distance from New Patient', distances)

sorted_by_distance = ckd_with_distances.sort('Distance from New Patient')

sorted_by_distance
```

Out[33]:

Class	Age	Sodium	Color	status	Distance from New Patient
0	0.739614	-0.247546	gold	CKD free	0.0604358
1	0.804284	-0.381492	blue	CKD	0.131562
1	0.933625	-0.247546	blue	CKD	0.133647
0	0.804284	-0.1136	gold	CKD free	0.136467
1	0.674944	-0.381492	blue	CKD	0.181464
1	0.610274	-0.247546	blue	CKD	0.189742
1	0.610274	-0.1136	blue	CKD	0.233669
1	0.739614	-0.515439	blue	CKD	0.272221
0	0.674944	0.0203463	gold	CKD free	0.297869
1	0.610274	-0.515439	blue	CKD	0.326272

... (148 rows omitted)

In [34]:
```
patient_1_nn = sorted_by_distance.take(np.arange(1))
patient_1_nn
```

Out[34]:

Class	Age	Sodium	Color	status	Distance from New Patient
0	0.739614	-0.247546	gold	CKD free	0.0604358

d)

In [35]:
```
patient_3_nn = sorted_by_distance.take(np.arange(3))
patient_3_nn
```

Out[35]:

Class	Age	Sodium	Color	status	Distance from New Patient
0	0.739614	-0.247546	gold	CKD free	0.0604358
1	0.804284	-0.381492	blue	CKD	0.131562
1	0.933625	-0.247546	blue	CKD	0.133647

e)

In [36]:
```
patient_9_nn = sorted_by_distance.take(np.arange(9))
patient_9_nn
```

Out[36]:

Class	Age	Sodium	Color	status	Distance from New Patient
0	0.739614	-0.247546	gold	CKD free	0.0604358
1	0.804284	-0.381492	blue	CKD	0.131562
1	0.933625	-0.247546	blue	CKD	0.133647
0	0.804284	-0.1136	gold	CKD free	0.136467
1	0.674944	-0.381492	blue	CKD	0.181464
1	0.610274	-0.247546	blue	CKD	0.189742
1	0.610274	-0.1136	blue	CKD	0.233669
1	0.739614	-0.515439	blue	CKD	0.272221
0	0.674944	0.0203463	gold	CKD free	0.297869

f)

In [37]:
```
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

def all_distances(training, new_point):
    attributes = training.drop('Class')
    def distance_from_point(row):
        return distance(np.array(new_point), np.array(row))
    return attributes.apply(distance_from_point)

def table_with_distances(training, new_point):
    return training.with_column('Distance', all_distances(training, new_point))

def closest(training, new_point, k):
    with_dists = table_with_distances(training, new_point)
    sorted_by_distance = with_dists.sort('Distance')
    topk = sorted_by_distance.take(np.arange(k))
    return topk

def majority(topkclasses):
    ones = topkclasses.where('Class', are.equal_to(1)).num_rows
    zeros = topkclasses.where('Class', are.equal_to(0)).num_rows
    if ones > zeros:
        return 1
    else:
        return 0

def classify(training, new_point, k):
    closestk = closest(training, new_point, k)
    topkclasses = closestk.select('Class')
    return majority(topkclasses)
```

In [39]:
```
ckd_small_p = Table().with_columns(
    'Age', standard_units(ckd.column('Age')),
    'Sodium', standard_units(ckd.column('Sodium')),
    'Potassium', standard_units(ckd.column('Potassium')),
    'Class', ckd.column('Class')
)

nrow = ckd_small_p.num_rows

shuffled_ckd = ckd_small_p.sample(with_replacement=False)
training_set = shuffled_ckd.take(np.arange(int(nrow/2)))
test_set = shuffled_ckd.take(np.arange(int(nrow/2), int(nrow)))
```

In [40]:
```
def count_zero(array):
    return len(array) - np.count_nonzero(array)

def count_equal(array1, array2):
    return count_zero(array1 - array2)

def evaluate_accuracy(training, test, k):
    test_attributes = test.drop('Class')
    def classify_testrow(row):
        return classify(training, row, k)
    c = test_attributes.apply(classify_testrow)
    print(count_equal(c, test.column('Class')), "correct predictions out of", test.num_rows, "total predictions")
    return count_equal(c, test.column('Class')) / test.num_rows
```

In [41]:
```
test_attributes = test_set.drop('Class')

def classify_testrow(row):
        return classify(training_set, row, 5)
predict_out = test_attributes.apply(classify_testrow)
actual_class= test_set.select("Class")

mod_pred = Table().with_columns('model prediction', predict_out)
table_compare =  mod_pred.append_column('actual class', actual_class[0])

table_compare
```

Out[41]:

model prediction	actual class
0	0
1	1
0	0
0	0
0	1
0	0
0	0
0	0
0	0
1	1

... (69 rows omitted)

In [42]:
```
evaluate_accuracy(training_set, test_set, 5)
```

```
68 correct predictions out of 79 total predictions
```

Out[42]:
```
0.8607594936708861
```

Knn classifier made 68 correct predictions out of 79 total predictions.

g) The accuracy is 0.8607594936708861.

h)

In [46]:
```
uniqueValues, occurCount = np.unique(training_set["Class"], return_counts=True)

print("Unique Values : " , uniqueValues)
print("Count : ", occurCount)
```

```
Unique Values :  [0 1]
Count :  [58 21]
```

In [47]:
```
uniqueValues_test, occurCount_test = np.unique(test_set["Class"], return_counts=True)

print("Unique Values (test) : " , uniqueValues_test)
print("Count (test) : ", occurCount_test)

npredictions = len(test_set["Class"])
modal_guess_accuracy = occurCount_test[0] / npredictions

print("\nmodal guess accuracy is", modal_guess_accuracy)
```

```
Unique Values (test) :  [0 1]
Count (test) :  [57 22]

modal guess accuracy is 0.7215189873417721
```

3. a)

```
In [49]: world
```

Out[49]:

lifeex_total	gini10	dem_score14	oil	gender_unequal	gdppcap08	literacy	pop_urban	country
45.02	29.4	2.77	0	0.797	1588	28.1	22.6	Afghanistan
77.41	33	5.67	5400	0.545	7715	89.9	51.9	Albania
74.5	35.3	3.83	2.125e+06	0.594	8033	69.9	66.5	Algeria
38.76	58.6	3.35	1.948e+06	0.756	5899	67.4	58.5	Angola
76.95	48.8	6.84	796300	0.534	14333	97.2	92.4	Argentina
73.23	30.2	4.13	0	0.57	6070	99.4	64.2	Armenia
81.81	35.2	9.01	589200	0.296	35677	99	89.1	Australia
79.78	29.1	8.54	21880	0.3	38152	98	67.6	Austria
67.36	36	2.83	1.011e+06	0.553	8765	98.8	51.9	Azerbaijan
78.15	29	2.87	48560	0.512	34605	86.5	88.6	Bahrain

... (156 rows omitted)

Albania is the second country in the data

b)

```
In [48]: world = Table.read_table('world_data.csv')


nrow_world = world.num_rows

train, test = world.split(int(nrow_world/2))
print(train.num_rows, 'training and', test.num_rows, 'test instances.')
```

```
83 training and 83 test instances.
```

There are 83 observations in each.

c)

```
In [52]: train_df =train.to_df()

y = train_df[["lifeex_total"]]
X = train_df[["dem_score14","oil","gender_unequal","gdppcap08"]]

Xc = sm.add_constant(X)
est = sm.OLS(y, Xc).fit()
print(est.summary())
```

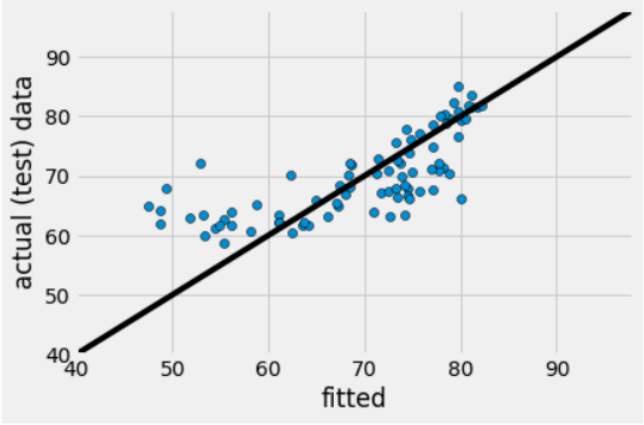
```
                            OLS Regression Results
==============================================================================
Dep. Variable:           lifeex_total   R-squared:                       0.519
Model:                            OLS   Adj. R-squared:                  0.494
Method:                 Least Squares   F-statistic:                     21.05
Date:                Mon, 11 Nov 2019   Prob (F-statistic):           8.44e-12
Time:                        00:19:58   Log-Likelihood:                -278.80
No. Observations:                  83   AIC:                             567.6
Df Residuals:                      78   BIC:                             579.7
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         83.3249      6.507     12.805      0.000      70.370      96.280
dem_score14    0.4551      0.498      0.913      0.364      -0.537       1.447
oil         3.764e-07   4.38e-07      0.858      0.393   -4.96e-07    1.25e-06
gender_unequal -31.2900      7.098     -4.408      0.000     -45.421     -17.159
gdppcap08   7.794e-05   6.69e-05      1.165      0.247   -5.52e-05       0.000
==============================================================================
Omnibus:                        7.796   Durbin-Watson:                   2.295
Prob(Omnibus):                  0.020   Jarque-Bera (JB):                7.307
Skew:                          -0.692   Prob(JB):                       0.0259
Kurtosis:                       3.441   Cond. No.                     2.48e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.48e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

d) For each unit increase in GDP per capita, predict life expectancy in years would increase by 7.794e-05.

e)

In [55]:
```
test_df =test.to_df()
X_test = test_df[["dem_score14","oil","gender_unequal","gdppcap08"]]

Xnew = sm.add_constant(X_test)
ynew =  est.predict(Xnew)
```

In [56]:
```
plots.scatter(test_df['lifeex_total']],ynew, edgecolor="black")
plots.ylabel("actual (test) data")
plots.xlabel("fitted")
plots.xlim(40,98)
plots.ylim(40,98)
plots.plot([-100, 5e5], [-100, 5e5], color="black")
plots.show()
```



The model does well between 60-80 years LE, but not so well between 40 to 60.

f)

In [57]:
```
iMSE = np.mean((ynew - test_df['lifeex_total'])**2)
print("\n the regression (w intercept) RMSE was:", (iMSE)**.5, "years")
```

```
 the regression (w intercept) RMSE was: 6.447613950089219 years
```

g)

In [58]:
```
def distance(pt1, pt2):
    return np.sqrt(sum((pt1 - pt2) ** 2))

def row_distance(row1, row2):
    return distance(np.array(row1), np.array(row2))

def distances(training, example, output):
    dists = []
    attributes = training.drop(output)
    for row in attributes.rows:
        dists.append(row_distance(row, example))
    return training.with_column('Distance', dists)

def closest(training, example, k, output):
    return distances(training, example, output).sort('Distance').take(np.arange(k))


def predict_nn(example):
    return np.average(closest(train2, example, 5, 'lifeex_total').column('lifeex_total'))
```

In [59]:
```
train2 = train.select("lifeex_total","dem_score14","oil","gender_unequal","gdppcap08")
test2 = test.select("dem_score14","oil","gender_unequal","gdppcap08")

nn_test_predictions = test2.apply(predict_nn)
```

In [60]:
```
rmse_nn = np.mean((test.column("lifeex_total") - nn_test_predictions) ** 2) ** 0.5
print("\n RMSE of the nn model is:", rmse_nn, "years")
```

```
 RMSE of the nn model is: 7.702675438811028 years
```

It fits worse than the regression.

4.

a) In general, we expect the probability of a positive test result given the disease to be higher than the probability of having the disease given a posiive test result.

b) Pr(D|test+)= Pr(test+|D)* Pr(D)/ Pr(test+)=0.95*0.001/0.02=0.475.

c) Pr(not D|test+)=1- Pr(D|test+)=0.525

5. a)

In [50]:
```
import numpy as np
import pandas as pd
```

In [54]:
```
url='https://raw.githubusercontent.com/fortunedatateam/f500-diversity/master/2017-f500-diversity-data.csv'
#import data

pd.set_option('display.max_column',212)
#display full data set

data=pd.read_csv(url)
#read data

data[:10]
#select first ten rows
```

Out[54]:

	f500-2017-rank	name	data-avail	data-url	diversity-pg-url	data-year	PAYROLL_START	PAYROLL_EI
0	1	Wal-Mart Stores	Partial	https://cdn.corporate.walmart.com/8c/08/6bc1b6...	http://corporate.walmart.com/our-story/working...	2015.0	NaN	N
1	2	Berkshire Hathaway	N	NaN	NaN	NaN	NaN	N
2	3	Apple	Y	https://images.apple.com/diversity/pdf/2016-EE...	https://www.apple.com/diversity/	2016.0	7/1/2016	7/31/20
3	4	Exxon Mobil	N	NaN	http://corporate.exxonmobil.com/en/community/c...	NaN	NaN	N
4	5	McKesson	N	NaN	http://www.mckesson.com/about-mckesson/corpora...	NaN	NaN	N
5	6	UnitedHealth Group	N	NaN	http://www.unitedhealthgroup.com/About/Diversi...	NaN	NaN	N
6	7	CVS Health	N	NaN	https://cvshealth.com/about/diversity	NaN	NaN	N
7	8	General Motors	N	NaN	https://www.gm.com/company/diversity/featured-...	NaN	NaN	N
8	9	AT&T	N	NaN	http://about.att.com/sites/diversity	NaN	NaN	N
9	10	Ford Motor	N	NaN	https://corporate.ford.com/company/diversity.html	NaN	NaN	N

b)

In [76]:
```
data.rename(columns={'data-url':'url'},inplace=True)#rename data-url to url
data.rename(columns={'diversity-pg-url':'diversity-url'},inplace=True)#rename diversity-pg-url to diversity-url
data[:3]#display the first three rows
```

Out[76]:

	f500-2017-rank	name	data-avail	url	diversity-url	data-year	PAYROLL_START	PAYROLL_END	HISPM1	HISF
0	1	Wal-Mart Stores	Partial	https://cdn.corporate.walmart.com/8c/08/6bc1b6...	http://corporate.walmart.com/our-story/working...	2015.0	NaN	NaN	NaN	
1	2	Berkshire Hathaway	N	NaN	NaN	NaN	NaN	NaN	NaN	
2	3	Apple	Y	https://images.apple.com/diversity/pdf/2016-EE...	https://www.apple.com/diversity/	2016.0	7/1/2016	7/31/2016	2.0	

c)

```
data2=data[['name','data-avail','data-year','HISPM1','HISPM1_2','HISPM2','HISPM3','HISPM4','HISPM5','HISPM6','HISPM7','HISPM8',
            'HISPM9','HISPM10','HISPM11','HISPF1','HISPF1_2','HISPF2','HISPF3','HISPF4','HISPF5','HISPF6','HISPF7','HISPF8','HISPF9',
            'HISPF10','HISPF11','TOTAL1','TOTAL1_2','TOTAL2','TOTAL3','TOTAL4','TOTAL5','TOTAL6','TOTAL7','TOTAL8','TOTAL9','TOTAL10',
            'TOTAL11']]
#display only the useful information

data2Y=data2[data2['data-avail']=='Y']#display only companies that shared their full data

data2Y.sort_values('data-year',ascending=False)#sort by year
data2Y[:5]#display the first five rows
```

	name	data-avail	data-year	HISPM1	HISPM1_2	HISPM2	HISPM3	HISPM4	HISPM5	HISPM6	HISPM7	HISPM8	HISPM9	HISPM10	HISPM11	HISPF1	HISF
2	Apple	Y	2016.0	2.0	411.0	799.0	1709.0	2562.0	672.0	13.0	0.0	0.0	132.0	6300.0	5774.0	0.0	
11	Amazon.com	Y	2016.0	1.0	489.0	790.0	117.0	27.0	246.0	32.0	612.0	10122.0	4.0	12440.0	8178.0	0.0	
15	Costco Wholesale	Y	2016.0	4.0	1401.0	149.0	292.0	3835.0	464.0	143.0	5937.0	6113.0	2968.0	21306.0	20354.0	0.0	
26	Alphabet	Y	2016.0	0.0	340.0	1107.0	29.0	81.0	25.0	0.0	0.0	0.0	52.0	1634.0	1216.0	0.0	
27	Microsoft	Y	2016.0	6.0	323.0	1937.0	0.0	287.0	1.0	0.0	0.0	0.0	0.0	2554.0	2356.0	0.0	

d)

```
In [118]: data2Y.shape
Out[118]: (16, 39)
```

There are 16 rows and 39 columns.

e)

```
In [119]: data2Y['percent_m']=data2Y["HISPM10"]/data2Y['TOTAL10']#create the new male Hispanic/Latina percentage column
          data2Y[:5]#display the first five rows
```

```
/share/apps/jupyterhub/2019-FA-DS-UA-111/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[119]:

?F8	HISPF9	HISPF10	HISPF11	TOTAL1	TOTAL1_2	TOTAL2	TOTAL3	TOTAL4	TOTAL5	TOTAL6	TOTAL7	TOTAL8	TOTAL9	TOTAL10	TOTAL11	percent_m
0.0	97.0	3171.0	2746.0	107.0	8205.0	23200.0	16972.0	21315.0	4996.0	28.0	0.0	0.0	418.0	77192.0	72494.0	0.081615
9.0	1.0	10354.0	6248.0	105.0	13360.0	30443.0	1753.0	965.0	7712.0	432.0	6491.0	113247.0	33.0	174541.0	114158.0	0.071273
7.0	2964.0	16144.0	15327.0	39.0	11271.0	5141.0	5980.0	32853.0	7098.0	579.0	22099.0	38627.0	18664.0	142351.0	137262.0	0.149672
0.0	31.0	776.0	566.0	31.0	10184.0	32092.0	560.0	2251.0	939.0	0.0	0.0	0.0	276.0	41200.0	34304.0	0.039660
0.0	0.0	951.0	831.0	157.0	9665.0	49299.0	0.0	4429.0	707.0	0.0	0.0	0.0	0.0	51374.0	47986.0	0.049714

f)

```
In [120]: data2Y['percent_f']=data2Y["HISPF10"]/data2Y['TOTAL10']#create the new female Hispanic/Latina percentage column
          data2Y[:5]#display the first five rows
```

```
/share/apps/jupyterhub/2019-FA-DS-UA-111/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[120]:

9	HISPF10	HISPF11	TOTAL1	TOTAL1_2	TOTAL2	TOTAL3	TOTAL4	TOTAL5	TOTAL6	TOTAL7	TOTAL8	TOTAL9	TOTAL10	TOTAL11	percent_m	percent_f
0	3171.0	2746.0	107.0	8205.0	23200.0	16972.0	21315.0	4996.0	28.0	0.0	0.0	418.0	77192.0	72494.0	0.081615	0.041079
0	10354.0	6248.0	105.0	13360.0	30443.0	1753.0	965.0	7712.0	432.0	6491.0	113247.0	33.0	174541.0	114158.0	0.071273	0.059321
0	16144.0	15327.0	39.0	11271.0	5141.0	5980.0	32853.0	7098.0	579.0	22099.0	38627.0	18664.0	142351.0	137262.0	0.149672	0.113410
0	776.0	566.0	31.0	10184.0	32092.0	560.0	2251.0	939.0	0.0	0.0	0.0	276.0	41200.0	34304.0	0.039660	0.018835
0	951.0	831.0	157.0	9665.0	49299.0	0.0	4429.0	707.0	0.0	0.0	0.0	0.0	51374.0	47986.0	0.049714	0.018511

g)

```
In [113]: data2Y_m=data2Y.sort_values('percent_m',ascending=False).head(2)#sort by male hispanic percent and display only the top two companies
          data2Y_m[["name","percent_m"]]
```

Out[113]:

	name	percent_m
15	Costco Wholesale	0.149672
2	Apple	0.081615

```
In [114]: data2Y_f=data2Y.sort_values('percent_f',ascending=False).head(2)#sort by female hispanic percent and display only the top two companies
          data2Y_f[["name","percent_f"]]
```

Out[114]:

	name	percent_f
15	Costco Wholesale	0.113410
29	Citigroup	0.095131

Costco Wholesale and Apple has the highest percentage of male Hispanic/Latino employees, with basically 14.9672% for Costco Wholesale and 8.1615% for Apple. Costco Wholesale and Citigroup has the highest percentage of female Hispanic/Latino employees, with basically 11.3410% for Costco Wholesale and 9.5131% for Citigroup.

h) The two top employment levels should be 1-Senior OFF AND MGRS and 1_2 Mid OFF AND MGRS. Percentage of employees in these two levels best capture “leadership” because it displays Hispanic role in professional level.

i)

In [73]:
```
data2Y['percent_mF']=(data2Y["HISPM1"]+data2Y["HISPM1_2"])/(data2Y['TOTAL1']+data2Y['TOTAL1_2'])
#create the new male Hispanic/Latina PORF percentage column

data2Y_mF=data2Y.sort_values('percent_mF',ascending=False).head(1)
#sort by percent_mF and display only the top
data2Y_mF[["name","percent_mF"]]
#display only useful information
```

```
/share/apps/jupyterhub/2019-FA-DS-UA-111/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[73]:

	name	percent_mF
15	Costco Wholesale	0.124226

In [74]:
```
data2Y['percent_fF']=(data2Y["HISPF1"]+data2Y["HISPF1_2"])/(data2Y['TOTAL1']+data2Y['TOTAL1_2'])
#create the new female Hispanic/Latina PORF percentage column

data2Y_fF=data2Y.sort_values('percent_fF',ascending=False).head(1)
#sort by percent_fF and display only the top
data2Y_fF[["name","percent_fF"]]
#display only useful information
```

```
/share/apps/jupyterhub/2019-FA-DS-UA-111/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[74]:

	name	percent_fF
29	Citigroup	0.055139

Costco Wholesale has the highest representation Hispanic/Latino male in leadership with a percentage of basically 12.4226%. Citigroup has the highest representation Hispanic/Latino female in leadership with a percentage of basically 5.5139%.

j) Selection error and statistical error might influence. Selection error shows the sample we choose might not be representative for the true population. Statistical error shows that there might be recorded incorrectly that might influence the result.

k) If I have unlimited resources, I could take those companies that I currently don’t have enough data into account. I could study the population in stead of only companies that provide all data. The limitation of my method is it takes much more time to study the whole population instead of taking a sample.