Unit 12: Classification and ROC

Case Studies:

- To introduce the concept of **ROC and AUC Classifier Analysis** we will examine the relationship between a:
  - **Categorical response variable:** support for a certain opinion (favor/not in favor) and an
  - **Explanatory variables:**
    - Sex
    - Party, and
    - Age

Summary of Concepts:

- **Research questions we will be able to answer**
  - **Categorical Response + Some Explanatory Variables**
    - Is there a linear association between the interaction of two explanatory variables and a response variable?
- **Definitions/Properties**
  - Classifiers
  - True positive, true negative, false positive, false negative
  - Confusion matrices
  - Sensitivity = True Positive Rate
  - Specificity
  - False Positive Rate
  - ROC Curves
  - AUC
- **Modeling**
  - **Making 1/0 predictions with a classifier model:**
    - Use a logistic regression to ***predict/classify*** the 1/0 label for a given explanatory variable(s) value(s)
  - **Assessing the predictive power of a classifier model:**
    - Use ROC curves and AUC to assess the predictive power of a classification model.
  - **Picking the best threshold in a classifier model:**
    - Use ROC curves to pick the “best” predicted probability threshold p_0 to classify a point as either a 1 or a 0.

Unit 12: Classification and ROC Analysis

The previous section explored logistic regression as a tool for modeling the dependence of a binary response variable on other exogenous variables. Once a the model is built, it is possible to use it for prediction purposes, by viewing the logistic regression model as a **classification algorithm**. In this usage, the estimated probabilities (fitted values) from the logistic regression model provide scores that can be thresholded to create a 0/1 classfier.

For example, suppose we have a model that provides a probability of rain each day. We might view a fitted probability greater than or equal to 50% as a prediction of rain, and a fitted probability less than 50% as a prediction of no rain.

How good is the classifer? The confusion matrix provides a way to keep track of the performance by cross classifying the true and predicted classes. From the confusion matrix we can compute:

- **Sensitivity:** fraction of true positives that are called positive by the algorithm for a given threshold;
- **Specificity:** fraction of true negatives that are called negative by the algorithm for a given threshold.

The **ROC curve** provides an overall summary of how good a scoring system across all possible thresholds, by graphing sensitivity versus 1 - specificity. This can also be used to find the optimal threshold.

New package: scikit-learn - machine learning package

To install this on your computer enter the following command from a terminal or anaconda window:

```
conda install scikit-learn
```

Topic 1: Classification with Logistic Regression (with Simulated Data)

First let's generate 100 data points $(x_1, y_1), \ldots, (x_{100}, y_{100})$.

- y_i = Categorical Response Variable Value (1 or 0)
- x_i = Numerical Explanatory Variable (x_i are observations drawn from the standard normal distribution).
- Furthermore, let's create a relationship between x_i and y_i as follows:
  - $log - odds_i = log(p_i/(1-p_i)) = -0.7 + 2.1x_i$,
  - $odds_i = p_i/(1-p_i) = exp(-0.7 + 2.1x_i)$,
  - $y_i \sim Bernoulli(p_i)$

In [1]:

```
import numpy as np
import pandas as pd
import zipfile as zp
```

In [2]:

```
from scipy.stats import norm, bernoulli
```

In [3]:

```python
# set the coefficient values
b0, b1 = -0.7, 2.1
#
# generate exogenous variable
x = norm.rvs(size=100, random_state=12347)
print()
print('Simulated Explanatory Variable Values')
print(x)
#
# odds depend on x
print()
print('Predicted Odds Values (given these explanatory variable values)')
odds = np.exp(b0 + b1*x)
print(odds)
#
print()
print('Predicted Probabilties (given these explanatory variable values)'
print(odds/(1+odds))
# convert odds to probabilities, generate response y
print()
print('Randomly Generated 0/1 Response Variable Value (given the predicte
y = bernoulli.rvs(p=odds/(1+odds), size=100, random_state=1)
print(y)
dat = pd.DataFrame({'x':x, 'y':y})
print('Simulated Explanatory and Response Variables Data')
print(dat.head(10))
```

```
Simulated Explanatory Variable Values
[ 3.43686856e-01  1.84840043e+00  2.24359353e-01 -1.63366001e+00
  1.24553764e+00  1.71281180e+00 -6.87918329e-01 -1.18623852e+00
 -4.00249494e-01 -3.03626185e-01 -1.83256545e+00  1.74946221e+00
  1.42782655e-01  1.23757615e+00  1.53648398e+00  1.20246058e+00
 -4.03077542e-01  1.47360612e+00 -1.37076419e+00  3.41535013e-01
  3.33221347e-01 -6.07517496e-01 -1.69986595e+00  1.53454538e+00
  1.40225648e+00 -1.12625332e-01  1.16304827e+00 -8.78694596e-01
 -3.20791229e-01  2.55685889e-01  1.07326071e+00 -1.48523861e-01
  8.11851470e-02 -1.91191247e+00  8.56759066e-01 -4.51892464e-01
  4.76712331e-01  1.16111637e+00  2.04491326e+00 -7.58926671e-01
  5.79516051e-01  9.69182863e-01  7.10046985e-01  1.10390116e-02
  5.47076489e-01 -6.88079368e-01 -1.78661786e-01  8.02021891e-01
  4.29862250e-01  1.51651818e+00 -7.93327375e-01  6.73740653e-01
  1.52595028e+00  6.96695737e-01  1.41802423e+00 -2.25458753e-01
  3.59542591e-01  1.68028205e-01 -3.14180863e+00  2.64335071e-01
  2.75880372e-01 -2.26135949e+00  1.04048864e+00  1.67862105e-01
 -5.90468085e-01 -2.17562648e-03  1.34273589e+00 -1.36517281e+00
 -3.61178435e-01 -9.30398417e-01 -1.58673585e+00  6.56937274e-02
 -3.00638810e-01  9.26401586e-01  8.33788612e-01 -1.32009183e+00
 -6.75361967e-02 -1.24408261e-01  1.89410149e-01 -7.06132732e-01
 -1.91851061e+00 -4.62411156e-01  9.61616777e-01 -1.85034522e-01
 -1.63128816e+00  4.30338583e-01 -9.24596753e-01  8.00132265e-01
  1.07029507e-01 -1.15224568e+00 -2.33536473e+00 -8.35873539e-01
 -3.27614542e-01  2.91775470e-01  1.43076788e+00  2.27007007e+00
  1.59071424e+00 -1.06840685e+00 -2.82712742e+00 -8.08333385e-01]

Predicted Odds Values (given these explanatory variable values)
[1.02198049e+00 2.40862442e+01 7.95451534e-01 1.60718495e-02
```

```
 6.79120948e+00 1.81179795e+01 1.17111537e-01 4.11265703e-02
 2.14268809e-01 2.62470919e-01 1.05842583e-02 1.95675120e+01
 6.70215200e-01 6.67861020e+00 1.25111013e+01 6.20383234e+00
 2.13000059e-01 1.09635141e+01 2.79146176e-02 1.01737271e+00
 9.99764856e-01 1.38652186e-01 1.39857198e-02 1.24602715e+01
 9.43794814e+00 3.91992255e-01 5.71104132e+00 7.84527574e-02
 2.53178228e-01 8.49540526e-01 4.72963244e+00 3.63527581e-01
 5.88892755e-01 8.95972109e-03 3.00174576e+00 1.92247019e-01
 1.35133892e+00 5.68791862e+00 3.63908672e+01 1.00887826e-01
 1.67696180e+00 3.80107534e+00 2.20581856e+00 5.08231577e-01
 1.56652631e+00 1.17071939e-01 3.41233000e-01 2.67579350e+00
 1.22471814e+00 1.19973787e+01 9.38565724e-02 2.04389106e+00
 1.22373847e+01 2.14483162e+00 9.75569170e+00 3.09293882e-01
 1.05658229e+00 7.06705839e-01 6.77027861e-04 8.65111957e-01
 8.86343045e-01 4.30122708e-03 4.41508082e+00 7.06459375e-01
 1.43706396e-01 4.94321672e-01 8.32901633e+00 2.82443201e-02
 2.32590772e-01 7.03806665e-02 1.77362396e-02 5.70043177e-01
 2.64122702e-01 3.47447480e+00 2.86038459e+00 3.10488707e-02
 4.30922934e-01 3.82411759e-01 7.39161616e-01 1.12716582e-01
 8.83643051e-03 1.88046985e-01 3.74115816e+00 3.36696787e-01
 1.61521012e-02 1.22594384e+00 7.12433939e-02 2.66519641e+00
 6.21737043e-01 4.41697141e-02 3.68212246e-03 8.58344848e-02
 2.49576315e-01 9.16428239e-01 1.00202951e+01 5.83901465e+01
 1.40202107e+01 5.26728895e-02 1.31099671e-03 9.09450221e-02]

Predicted Probabilties (given these explanatory variable values)
[5.05435385e-01 9.60137516e-01 4.43037040e-01 1.58176309e-02
 8.71650223e-01 9.47693217e-01 1.04834238e-01 3.95019890e-02
 1.76459123e-01 2.07902547e-01 1.04734051e-02 9.51379632e-01
 4.01274758e-01 8.69768099e-01 9.25986789e-01 8.61184998e-01
 1.75597732e-01 9.16412519e-01 2.71565528e-02 5.04305775e-01
 4.99941207e-01 1.21768691e-01 1.37928173e-02 9.25707294e-01
 9.04195730e-01 2.81605198e-01 8.50991828e-01 7.27456598e-02
 2.02028907e-01 4.59325175e-01 8.25468735e-01 2.66608161e-01
 3.70630902e-01 8.88015736e-03 7.50109062e-01 1.61247641e-01
 5.74710396e-01 8.50476649e-01 9.73255501e-01 9.16422400e-02
 6.26442186e-01 7.91713329e-01 6.88067187e-01 3.36971845e-01
 6.10368304e-01 1.04802506e-01 2.54417391e-01 7.27949897e-01
 5.50504856e-01 9.23061409e-01 8.58033629e-02 6.71473130e-01
 9.24456377e-01 6.82017951e-01 9.07025970e-01 2.36229533e-01
 5.13756388e-01 4.14075948e-01 6.76569804e-04 4.63839157e-01
 4.69873731e-01 4.28280576e-03 8.15330549e-01 4.13991324e-01
 1.25649726e-01 3.30800042e-01 8.92807562e-01 2.74684912e-02
 1.88700725e-01 6.57529314e-02 1.74271476e-02 3.63074841e-01
 2.08937552e-01 7.76510083e-01 7.40958452e-01 3.01138690e-02
 3.01150344e-01 2.76626523e-01 4.25010309e-01 1.01298555e-01
 8.75903193e-03 1.58282448e-01 7.89081071e-01 2.51887182e-01
 1.58953578e-02 5.50752367e-01 6.65053286e-02 7.27163326e-01
 3.83377222e-01 4.23012787e-02 3.66861418e-03 7.90493266e-02
 1.99728750e-01 4.78195959e-01 9.09258329e-01 9.83162190e-01
 9.33423038e-01 5.00372813e-02 1.30928025e-03 8.33635245e-02]

Randomly Generated 0/1 Response Variable Value (given the predicted proba
bilities)
[1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 1 0
0
 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0
```

```
0
 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0]
Simulated Explanatory and Response Variables Data
          x  y
0  0.343687  1
1  1.848400  1
2  0.224359  0
3 -1.633660  0
4  1.245538  1
5  1.712812  1
6 -0.687918  0
7 -1.186239  0
8 -0.400249  0
9 -0.303626  0
```

Next, let's fit the simulated data with a model.

In [4]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [5]:
```
simmod = smf.logit('y ~ x', data=dat).fit()
simmod.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.403715
         Iterations 7
```

Out[5]:

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	100
Model:	Logit	**Df Residuals:**	98
Method:	MLE	**Df Model:**	1
Date:	Wed, 04 Nov 2020	**Pseudo R-squ.:**	0.4149
Time:	19:08:31	**Log-Likelihood:**	-40.371
converged:	True	**LL-Null:**	-68.994
Covariance Type:	nonrobust	**LLR p-value:**	3.846e-14

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-0.4938	0.291	-1.697	0.090	-1.064	0.076
x	2.2133	0.440	5.028	0.000	1.350	3.076

Then, let's make predictions for the p_i's based on the model and some *new* x_i values. These are the predictive probabilities.

```
In [6]:  # example predictive probabilities
         simmod.predict(exog=dict(x=[-2,-1,0,1,2]))
```

```
Out[6]: 0    0.007243
        1    0.062554
        2    0.378988
        3    0.848056
        4    0.980786
        dtype: float64
```

Then, let's plot:

* 100 simulated $(x_1, y_1), \ldots, (x_{100}, y_{100})$ points and

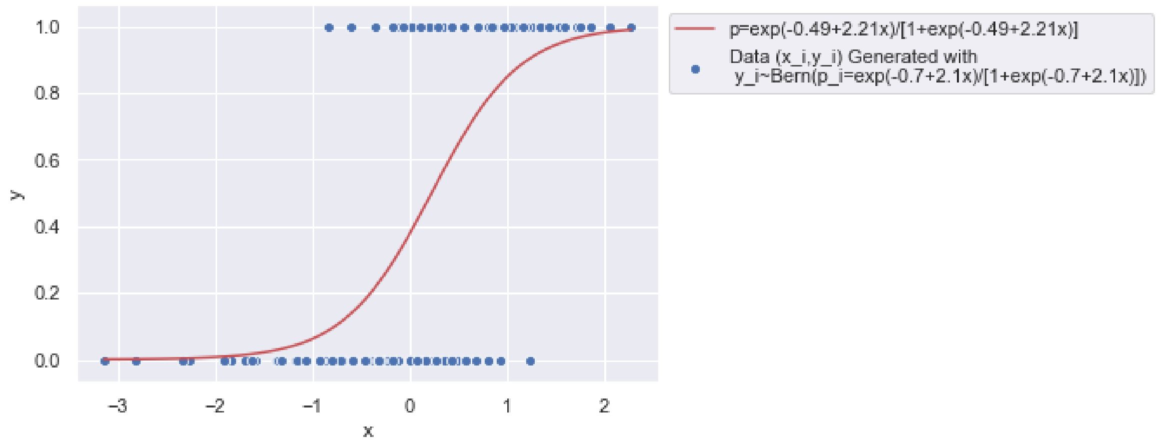
* the curve that represents the model's predicted probability p for any given value of x.

```
In [7]:  import matplotlib.pyplot as plt
         import seaborn as sns; sns.set()
```

```
Bad key "text.kerning_factor" on line 4 in
C:\Users\vme3\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotli
b\mpl-data\stylelib\_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
 (https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templat
e)
or from the matplotlib source distribution
```

In [8]:

```python
# scatter plot of raw 0/1 data
sns.scatterplot(x='x', y='y', data=dat, label='Data (x_i,y_i) Generated w
#
# make a grid of x values for the probability curve
xgrid = np.linspace(dat['x'].min(), dat['x'].max(), 100)
#
# get predictive probabilities for the grid
pgrid = simmod.predict(exog=dict(x=xgrid))
#
# add the curve to the plot
plt.plot(xgrid, pgrid, color='r', label='p=exp(-0.49+2.21x)/[1+exp(-0.49
plt.legend(bbox_to_anchor=(1,1))
plt.show()
```



Interpret this relationship between the curve and the data:

What does this model $p = exp(-0.49 + 2.21x)/[1 + exp(-0.49 + 2.21x)]$ actually predict for a given value of x?

How can we use this model $p = exp(-0.49 + 2.21x)/[1 + exp(-0.49 + 2.21x)]$ to CLASSIFY a value of x as having either a y=0 or y=1 response variable value?

What happens is this threshold is $p_0 = 0.5$?

In [9]:

```python
pred_probabilities=simmod.predict(exog=dict(x=dat.x))
dat['pred']=pred_probabilities
dat
```

Out[9]:

	x	y	pred
0	0.343687	1	0.566315
1	1.848400	1	0.973330
2	0.224359	0	0.500681
3	-1.633660	0	0.016149
4	1.245538	1	0.905756
...	...	...	...
95	2.270070	1	0.989339
96	1.590714	1	0.953773
97	-1.068407	0	0.054242
98	-2.827127	0	0.001168
99	-0.808333	0	0.092548

100 rows × 3 columns

Will there exist a threshold value of p_0 that will create no misclassifications of the given data?

Sketch an example of a dataset and predictive probability curve in which there *would* exist a threshold that would create no misclassifications of the given data.

Topic 2: Choosing classification thresholds (for logistic regression)?

One way: Classification threshold derived from predictive probability p (for a given x that has been plugged into the model)

How accurate is this model as a classifier? It depends in part on where we set the **threshold** p_0.

Let $\hat{p}(x)$ denote the estimated probability as a function of x. Suppose our classification rule is $I(\hat{p} > p_0)$ where $I(True) = 1$ and $I(False) = 0$.

Another way (for simple logistic regression): Classification threshold in terms of x that has been *derived* from predictive probability

Because the fitted probability function is increasing as a function of x, the probability threshold is equivalent to a threshold for x.

First, the **logit of the threshold** p_0 is

$$\log\left(\frac{p_0}{1-p_0}\right).$$

Furthermore, the **fitted model** has the form

$$\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{\beta}_0 + \hat{\beta}_1 x$$

Classifier Threshold (in terms of x)

Setting this equal to the threshold logit and solving for x gives the classifier threshold:

$$x_0 = \frac{\text{logit}(p_0)-\hat{\beta}_0}{\hat{\beta}_1}$$

Theory/Relationship for a Common Threshold $p_0 = 0.5$

If we set $p_0 = 0.5$, a common default, then $\text{logit}(p_0) = 0$ and the threshold is simply $x_0 = -\frac{\hat{\beta}_0}{\hat{\beta}_1}$.

Topic 3: Exploring Different Classification Thresholds

In [10]:
```
simmod.params
```
Out[10]:
```
Intercept   -0.493845
x            2.213281
dtype: float64
```

Let's first use $p_0 = 0.5$

In [11]:
```
#### Set probability threshold
pthresh = 0.50
```

Thus the corresponding x_0 threshold is:

In [12]:
```
#### Compute corresponding x threshold
xthresh = (np.log(pthresh/(1-pthresh)) \
              -simmod.params[0])/simmod.params[1]
xthresh
```
Out[12]:
```
0.22312813097265544
```

Let's make our y predictions using this threshold.

In [13]:

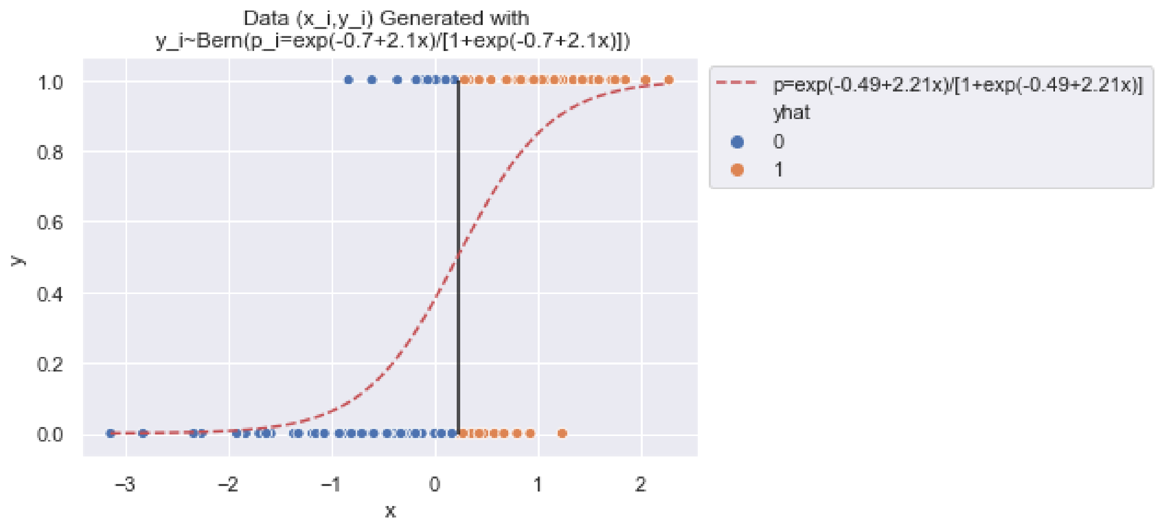
```
#### Predicted y based on threshold
dat['yhat'] = 1*(dat['x'] >= xthresh)
dat.head()
```

Out[13]:

	x	y	pred	yhat
0	0.343687	1	0.566315	1
1	1.848400	1	0.973330	1
2	0.224359	0	0.500681	1
3	-1.633660	0	0.016149	0
4	1.245538	1	0.905756	1

In [14]:

```
# scatter plot of raw 0/1 data
sns.scatterplot(x='x', y='y', hue='yhat', data=dat)
#
# make a grid of x values for probability curve
xgrid = np.linspace(dat['x'].min(), dat['x'].max(), 100)
#
# get predictive probabilities for the grid
pgrid = simmod.predict(exog=dict(x=xgrid))
#
# add probability curve to the graph
plt.plot(xgrid, pgrid, color='r', linestyle='dashed',label='p=exp(-0.49+
#
# add x threshold line for 0/1 classification
plt.vlines(x=xthresh, ymin=0, ymax=1)
plt.legend(bbox_to_anchor=(1,1))
plt.title('Data (x_i,y_i) Generated with \n y_i~Bern(p_i=exp(-0.7+2.1x)/
plt.show()
```



Topic 4: How well did our classifer do with a given threshold p_0 (or equivalently x_0)?

How is the performance of the classifier on the training data? Because the 0's and 1's have overlapping x values, no classifier based on x only can perfectly classify them.

Definitions

Orange = Predicted Positive (predicted to be 1)

True Positive:

False Positive:

Blue = Predicted Negative (predicted to be 0)

True Negative:

False Negative:

- The orange colored observations with $y = 1$ are correctly classified because they are above the x threshold and therefore have $\hat{y} = 1 = y$.
- The blue colored observations with $y = 0$ are correctly classified because they are below the x threshold and therefore $\hat{y} = 0 = y$.

- The blue colored observations with $y = 1$ and the orange colored observations with $y = 0$ are incorrectly classified because $\hat{y} \neq y$.

Confusion matrix for a given threshold.

Calculated with:

1. the *true* values (positive = 1 and negative = 0)
2. the *predicted* values (positive = 1 and negative = 0) using a given threshold (p_0 or x_0)

We can summarize the classification performance for a given classifier by comparing the predicted classes to the true classes. The we cross classify the results.

Classification	Actual Negative (0)	Actual Positive (1)
Predicted Negative (0)	TN = True Neg	FN = False Neg
Predicted Positive (1)	FP = False Pos	TP = True Pos

In [15]:
```
# This import requires that you already
# installed the scikit-learn library
# as described in the introduction to this chapter.
#
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
```

In [16]:
```
confusion_matrix(y_true=dat['y'], y_pred=dat['yhat'])
```

Out[16]:
```
array([[44, 10],
       [10, 36]], dtype=int64)
```

For the given threshold of $p_0 = 0.5$ (or equivalently $x_0=0.223) what is the number of:

- true negatives?

- false positives?

- false negatives?

- true positives?

```
In [17]:  tn, fp, fn, tp = confusion_matrix(y_true=dat['y'],
                                            y_pred=dat['yhat']).ravel()
          (tn, fp, fn, tp)
Out[17]:  (44, 10, 10, 36)
```

Sensitivity Rate = True Positive Rate

```
In [18]:  # Sensitivity = true positive rate
          tp / (fn + tp)
Out[18]:  0.782608695652174
```

Specificity Rate

```
In [19]:  # Specificity = true negative rate
          tn / (fp + tn)
Out[19]:  0.8148148148148148
```

False Positive Rate = 1-Specificity Rate

In [20]:
```python
# False positive rate
fp / (fp + tn)
```
Out[20]:
```
0.18518518518518517
```

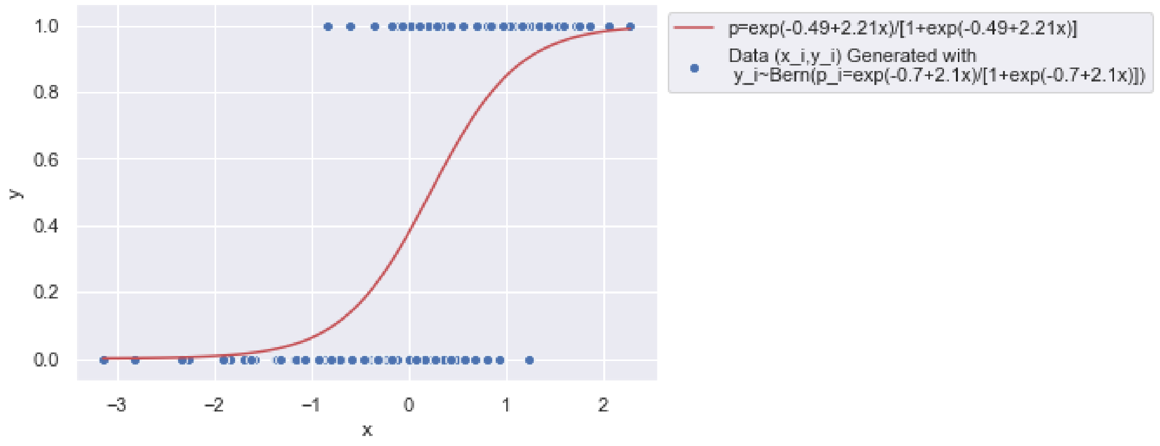
Relationship between Sensitivity and Specificity

What will happen to sensitivity and specificity if we move p_0 up (or equivalently move x_0 right)?

What will happen to sensitivity and specificity if we move p_0 down (or equivalently move x_0 left)?

In [21]:

```
# scatter plot of raw 0/1 data
sns.scatterplot(x='x', y='y', data=dat, label='Data (x_i,y_i) Generated w
#
# make a grid of x values for the probability curve
xgrid = np.linspace(dat['x'].min(), dat['x'].max(), 100)
#
# get predictive probabilities for the grid
pgrid = simmod.predict(exog=dict(x=xgrid))
#
# add the curve to the plot
plt.plot(xgrid, pgrid, color='r', label='p=exp(-0.49+2.21x)/[1+exp(-0.49
plt.legend(bbox_to_anchor=(1,1))
plt.show()
```



In general

Sensitivity and specificity are commonly used measures of classification performance. They play against each other in that we can increase one or the other of them by changing the classfiication threshold, but it will be at the cost of decreasing the other.

Topic 5: Taking into account all possible thresholds you could use, how do you assess how well your classifier model fit the data?

ROC Curve, plotting sensitivity and specificity across all thresholds.

How it's built:

The ROC curve provides a look at the inherent classification ability of a given scoring system, independently of where one sets the threshold on the scores. There will then be an optimal threshold based on the desired tradeoff, which may be deduced from the ROC curve.

Generating false positive rates and true positive rates for a list of p_0 thresholds

In [22]:

```python
fprs, tprs, thresholds = roc_curve(y_true=dat['y'],
                            y_score=simmod.fittedvalues)
auc = roc_auc_score(y_true=dat['y'],
                    y_score=simmod.fittedvalues)
print('False Positive Rates')
print(fprs)
print('True Positive Rates')
print(tprs)
print('p_0 Thresholds')
#The 'thresholds' output is a log_odds threshold. We want to convert eacl
# Remember odds= exp(log_odds)
# Also remember p = odds/(1+odds)
p_thresholds=np.exp(thresholds)/(1+np.exp(thresholds))
print(np.round(p_thresholds,2))
```

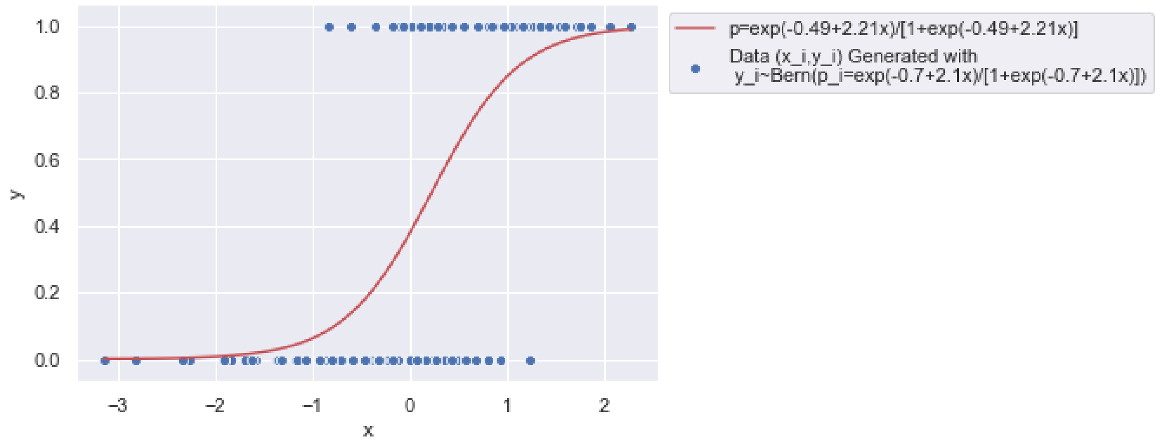
```
False Positive Rates
[0.         0.         0.         0.01851852 0.01851852 0.03703704
 0.03703704 0.05555556 0.05555556 0.09259259 0.09259259 0.12962963
 0.12962963 0.14814815 0.14814815 0.16666667 0.16666667 0.18518519
 0.18518519 0.24074074 0.24074074 0.25925926 0.25925926 0.27777778
 0.27777778 0.31481481 0.31481481 0.42592593 0.42592593 0.51851852
 0.51851852 0.62962963 0.62962963 1.        ]
True Positive Rates
[0.         0.02173913 0.34782609 0.34782609 0.5        0.5
 0.56521739 0.56521739 0.60869565 0.60869565 0.63043478 0.63043478
 0.65217391 0.65217391 0.73913043 0.73913043 0.7826087  0.7826087
 0.80434783 0.80434783 0.84782609 0.84782609 0.86956522 0.86956522
 0.91304348 0.91304348 0.93478261 0.93478261 0.95652174 0.95652174
 0.97826087 0.97826087 1.         1.        ]
p_0 Thresholds
[1.   0.99 0.91 0.9  0.84 0.83 0.78 0.78 0.74 0.69 0.67 0.61 0.61 0.57
 0.54 0.53 0.52 0.5  0.48 0.46 0.42 0.41 0.38 0.38 0.32 0.31 0.29 0.23
 0.22 0.14 0.14 0.09 0.09 0.   ]
```

Interpret the first elements in the three lists above

Interpret the last elements in the three lists above

In [23]:

```
# scatter plot of raw 0/1 data
sns.scatterplot(x='x', y='y', data=dat, label='Data (x_i,y_i) Generated w
#
# make a grid of x values for the probability curve
xgrid = np.linspace(dat['x'].min(), dat['x'].max(), 100)
#
# get predictive probabilities for the grid
pgrid = simmod.predict(exog=dict(x=xgrid))
#
# add the curve to the plot
plt.plot(xgrid, pgrid, color='r', label='p=exp(-0.49+2.21x)/[1+exp(-0.49-
plt.legend(bbox_to_anchor=(1,1))
plt.show()
```



ROC CURVE: Plot the false positive rates and the true positive rates in a line plot

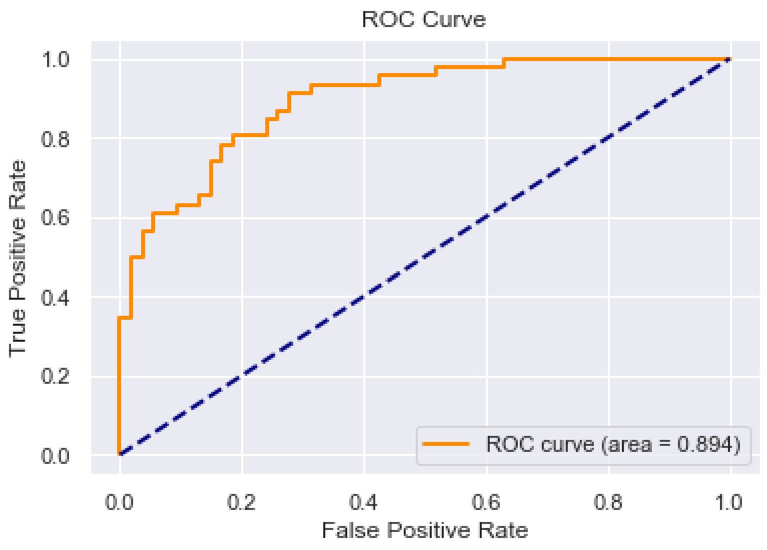
Let's define a function for plotting the ROC curve taking the arrays of false postive rates and true positive rates as arguments. (Modified from: https://stackabuse.com/understanding-roc-curves-with-python/ (https://stackabuse.com/understanding-roc-curves-with-python/))

In [24]:

```
def plot_roc(fpr, tpr, auc, lw=2):
    plt.plot(fpr, tpr, color='darkorange', lw=lw,
             label='ROC curve (area = '+str(round(auc,3))+')')
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()
```

In [25]:

```
plot_roc(fprs, tprs, auc)
```



For a given threshold for a model, what would be the BEST scenario for a (true positive rate, false positive rate) pair?

If our model *did* in fact have a threshold which produced the best possible scenario for a (true positive rate, false positive rate) pair, what would be the area under the whole ROC curve?

What is the "area under curve" (AUC) of the ROC curve that we have for this data?

In [26]:
```
auc = roc_auc_score(y_true=dat['y'],
                    y_score=simmod.fittedvalues)
auc
```

Out[26]:
```
0.8941223832528181
```

What is the WORST ROC curve that we could get (given any dataset and classification model trained on this dataset)?

Thus, what is the "area under curve" (AUC) of the WORST ROC curve that we could get (given any dataset and classification model trained on this dataset)?

Use the ROC Curve and AUC to assess how effective our overall classifier model is at predicting our response variable y.

Use the ROC Curve to pick out an "ideal" threshold for x_0.

ROC/AUC summary

The area under the curve (AUC) is one metric of the amount of information in a classification scoring system. It shows how the true positive and false positive rates track as we change the threshold in the scoring system from a maximum to a minimum possible threshold.

The curve captures the trade-off between **sensitivity (TP rate)** and **1 - specificity (1 - TN rate)** as the threshold changes.

The baselines for comparison are:

- Random guessing: its ROC curve is represented by the diagonal dashed line, and its AUC = 0.50.
- Perfect classification: Its ROC curve would jump up to a true positive rate of 1 at a false positive rate of 0; its ASUC = 1
- In our example, AUC = 0.89, which is pretty high depending on the context.

Example: Let's build a classifier to predict support for the border wall (given sex, age, and political party).

In [27]:
```
zf = zp.ZipFile('Feb17-public.zip')
missing_values = ["NaN", "nan", "Don't know/Refused (VOL.)"]
df = pd.read_csv(zf.open('Feb17public.csv'),
                 na_values=missing_values)[['age', 'sex', \
                                            'q52', 'party']]
```

Don't forget to translate your categorical response variable in 1's and 0's.

Remember...

- 1 = success = level you're interested in = favor border wall
- 0 = failure = level you're not interested in = not favor border wall

In [28]:
```
# reduce q52 responses to two categories and
# create binary reponse variable
df['q52'][df['q52']!='Favor'] = 'Not_favor'
df['y'] = df['q52'].map({'Not_favor':0,'Favor':1})
#
# use cleaned data without records that have missing values
dfclean = df.dropna()
dfclean.head(10)
```

Out[28]:

	age	sex	q52	party	y
0	80.0	Female	Not_favor	Independent	0
1	70.0	Female	Not_favor	Democrat	0
2	69.0	Female	Not_favor	Independent	0
3	50.0	Male	Favor	Republican	1
4	70.0	Female	Not_favor	Democrat	0
5	78.0	Male	Not_favor	Democrat	0
6	89.0	Female	Not_favor	Independent	0
7	92.0	Female	Not_favor	Republican	0
8	54.0	Female	Favor	Independent	1
9	58.0	Female	Not_favor	Independent	0

MODEL 1: Let's first model favor/not favor of border wall with just age and sex as explanatory variables.

In [29]:

```
mod1 = smf.logit('y ~ age + sex', data=dfclean).fit()
mod1.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.619057
         Iterations 5
```

Out[29]:

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	1465
Model:	Logit	**Df Residuals:**	1462
Method:	MLE	**Df Model:**	2
Date:	Wed, 04 Nov 2020	**Pseudo R-squ.:**	0.03557
Time:	19:08:34	**Log-Likelihood:**	-906.92
converged:	True	**LL-Null:**	-940.37
Covariance Type:	nonrobust	**LLR p-value:**	2.960e-15

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-2.0818	0.196	-10.637	0.000	-2.465	-1.698
sex[T.Male]	0.5415	0.114	4.750	0.000	0.318	0.765
age	0.0220	0.003	6.770	0.000	0.016	0.028

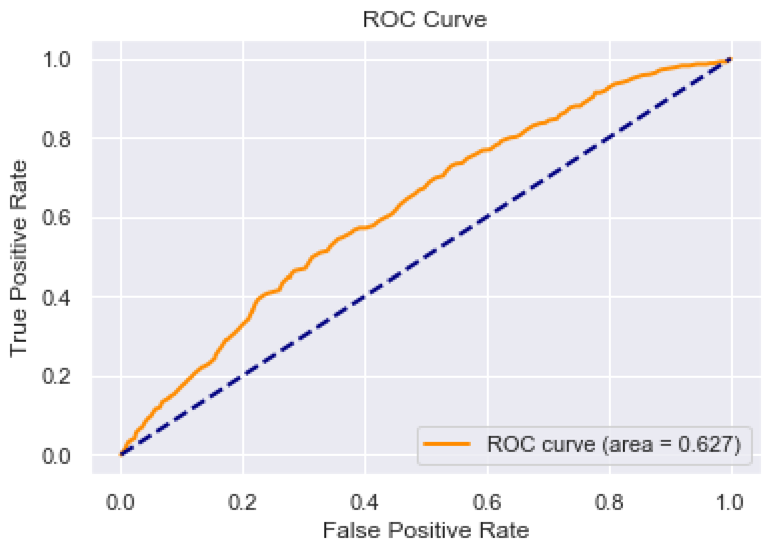
In [30]:

```
fprs, tprs, thresholds = roc_curve(y_true=dfclean['y'],
                                   y_score=mod1.fittedvalues)
auc = roc_auc_score(y_true=dfclean['y'],
                    y_score=mod1.fittedvalues)
print(auc)
```

```
0.6265544041450778
```

In [31]:

```
plot_roc(fprs, tprs, auc)
```



Assessing Effectiveness of Classifier Model 1

We see that although 'age' and 'sex' are statistically signficant variables in the model, using only these variables to predict 'y' is not much better than random guessing. There is too much variation beyond that explained by age and gender to rely on them for accurate classfication.

MODEL 2: Let's next see if we can make our model for predicting favor/not favor of border wall better if we NOW use the following explanatory variables:

- age
- sex
- political party

Let's see what happens if we add 'party' to the model as an additional explanatory variable.

In [32]:
```
mod2 = smf.logit('y ~ party + age + sex', data=dfclean).fit()
mod2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.466129
         Iterations 6
```

Out[32]:

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	1465
Model:	Logit	**Df Residuals:**	1458
Method:	MLE	**Df Model:**	6
Date:	Wed, 04 Nov 2020	**Pseudo R-squ.:**	0.2738
Time:	19:08:35	**Log-Likelihood:**	-682.88
converged:	True	**LL-Null:**	-940.37
Covariance Type:	nonrobust	**LLR p-value:**	4.971e-108

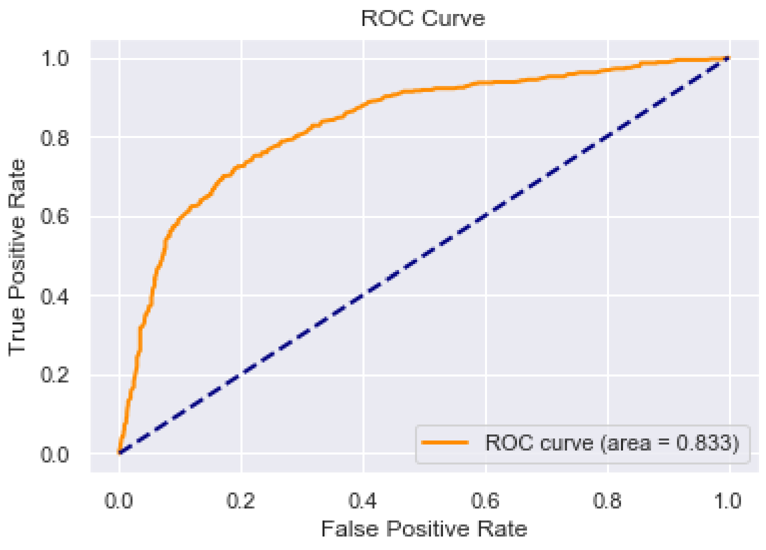
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-3.5261	0.281	-12.536	0.000	-4.077	-2.975
party[T.Independent]	1.6843	0.191	8.796	0.000	1.309	2.060
party[T.No preference (VOL.)]	1.8226	0.379	4.807	0.000	1.079	2.566
party[T.Other party (VOL.)]	2.8930	0.938	3.083	0.002	1.054	4.732
party[T.Republican]	3.5862	0.206	17.435	0.000	3.183	3.989
sex[T.Male]	0.3721	0.137	2.712	0.007	0.103	0.641
age	0.0168	0.004	4.305	0.000	0.009	0.024

Note that including 'party' entails that a few more of the rows have missing information than when we only included 'age' and 'sex'.

In [33]:
```
fprs, tprs, thresholds = roc_curve(y_true=dfclean['y'],
                                   y_score=mod2.fittedvalues)
auc = roc_auc_score(y_true=dfclean['y'],
                    y_score=mod2.fittedvalues)
print(auc)
```

```
0.8329699481865286
```

In [34]:
```
plot_roc(fprs, tprs, auc)
```



Assessing the Effectiveness of Classifier Model 2

It appears that including party affiliation in the model gives a big improvement in the ability of the model to predict the answer to 'q52', increasing the area under the curve from 0.63 to 0.83. There is still substantial variation remaining.

What threshold should we choose and how effective is our classifier with that threshold? What does that threshold mean?

We can see that the ROC curve starts to level off around a false positive rate around 0.42. This corresponds to a true positive rate of around 0.93. These rates were generated by using a p_0 threshold of 0.22.

We see that peoples' opinions are more than just the sum of their age, gender and party affiliation.

In [35]:

```python
print('False Positive Rates')
print(fprs)
print('True Positive Rates')
print(tprs)
print('p_0 Thresholds')
#The 'thresholds' output is a log_odds threshold. We want to convert eacl
# Remember odds= exp(log_odds)
# Also remember p = odds/(1+odds)
p_thresholds=np.exp(thresholds)/(1+np.exp(thresholds))
print(np.round(p_thresholds,2))
```

```
False Positive Rates
[0.         0.         0.         0.00103627 0.00103627 0.00207254
 0.00207254 0.00310881 0.00310881 0.00310881 0.00414508 0.00518135
 0.00621762 0.00725389 0.00725389 0.00725389 0.00829016 0.00829016
 0.00932642 0.01139896 0.01139896 0.01139896 0.01243523 0.01243523
 0.01243523 0.0134715  0.01450777 0.01450777 0.01865285 0.01865285
 0.01865285 0.01865285 0.01968912 0.02279793 0.0238342  0.0238342
 0.0238342  0.02487047 0.02487047 0.02694301 0.02797927 0.02901554
 0.02901554 0.02901554 0.03108808 0.03212435 0.03212435 0.03419689
 0.03419689 0.03419689 0.03419689 0.03419689 0.03419689 0.03419689
 0.03419689 0.0373057  0.03834197 0.04145078 0.04145078 0.04248705
 0.04248705 0.04455959 0.04455959 0.04663212 0.04766839 0.04870466
 0.04974093 0.05284974 0.05284974 0.05284974 0.05388601 0.05492228
 0.05492228 0.05699482 0.05803109 0.05803109 0.05803109 0.05906736
 0.05906736 0.05906736 0.05906736 0.0611399  0.0611399  0.06217617
 0.0642487  0.0642487  0.06632124 0.06735751 0.06735751 0.06839378
 0.06943005 0.07046632 0.07046632 0.07150259 0.07150259 0.07253886
 0.07357513 0.07357513 0.07357513 0.07357513 0.0746114  0.07564767
 0.07564767 0.07564767 0.07564767 0.07564767 0.07772021 0.07979275
 0.08082902 0.08290155 0.08290155 0.08290155 0.08290155 0.08393782
 0.08601036 0.08704663 0.0880829  0.0880829  0.08911917 0.08911917
 0.09119171 0.09222798 0.09430052 0.09533679 0.09637306 0.09637306
 0.10051813 0.10362694 0.10569948 0.10880829 0.10984456 0.1119171
 0.11502591 0.11502591 0.11606218 0.11606218 0.11917098 0.12020725
 0.12227979 0.12849741 0.13264249 0.13264249 0.13471503 0.14093264
 0.14404145 0.14507772 0.14611399 0.15025907 0.15129534 0.15129534
 0.15336788 0.15440415 0.15544041 0.15647668 0.15647668 0.15854922
 0.15854922 0.16165803 0.16165803 0.16580311 0.16683938 0.16787565
 0.16891192 0.16891192 0.17098446 0.17098446 0.17202073 0.18031088
 0.18238342 0.18445596 0.18549223 0.18549223 0.18756477 0.18963731
 0.19067358 0.19378238 0.19481865 0.19481865 0.20518135 0.20725389
 0.2134715  0.21761658 0.21761658 0.22072539 0.2238342  0.22487047
 0.22797927 0.23316062 0.23316062 0.23523316 0.23834197 0.24248705
 0.24352332 0.2507772  0.25595855 0.26217617 0.26217617 0.26321244
 0.26632124 0.26943005 0.27046632 0.27772021 0.28082902 0.28601036
 0.28911917 0.29119171 0.29222798 0.29326425 0.29533679 0.30259067
 0.30777202 0.31088083 0.31295337 0.31398964 0.31606218 0.31709845
 0.31813472 0.3253886  0.32642487 0.32849741 0.33160622 0.33264249
 0.34093264 0.34715026 0.34818653 0.35233161 0.35336788 0.35958549
 0.36165803 0.36787565 0.37202073 0.37305699 0.38134715 0.38341969
 0.38756477 0.38860104 0.39378238 0.39585492 0.39792746 0.4
 0.40207254 0.40518135 0.40725389 0.40932642 0.41243523 0.41658031
 0.41968912 0.42590674 0.43108808 0.43316062 0.4373057  0.44248705
 0.44766839 0.44870466 0.45284974 0.45388601 0.46010363 0.4611399
 0.46632124 0.47150259 0.4746114  0.47875648 0.48186528 0.48393782
 0.48497409 0.49119171 0.49430052 0.49533679 0.49637306 0.50259067
 0.50466321 0.50569948 0.51088083 0.51295337 0.51398964 0.51606218
 0.51813472 0.52020725 0.52331606 0.52746114 0.52953368 0.53678756
 0.54093264 0.5492228  0.55647668 0.55854922 0.5626943  0.56373057
 0.56580311 0.56683938 0.56994819 0.57202073 0.57823834 0.57927461
 0.58238342 0.58341969 0.5865285  0.58860104 0.59481865 0.59585492
 0.60207254 0.60310881 0.60829016 0.60932642 0.6134715  0.61450777
 0.61761658 0.62176166 0.62694301 0.63005181 0.63626943 0.63834197
 0.64145078 0.64559585 0.64870466 0.65803109 0.65906736 0.6642487
 0.67046632 0.68497409 0.69430052 0.69637306 0.70466321 0.70984456
 0.71088083 0.72124352 0.72642487 0.72746114 0.72953368 0.73678756
 0.74404145 0.74507772 0.75544041 0.76580311 0.77202073 0.77512953
 0.78134715 0.78756477 0.79067358 0.79792746 0.79896373 0.80310881
 0.80725389 0.81450777 0.81554404 0.81865285 0.82279793 0.83005181
 0.83108808 0.83834197 0.84248705 0.84559585 0.84663212 0.8507772
 0.85388601 0.85595855 0.85906736 0.8611399  0.86528497 0.87046632
 0.87357513 0.87875648 0.88186528 0.88290155 0.88393782 0.88911917
 0.89119171 0.89533679 0.89637306 0.90051813 0.90569948 0.91295337
 0.91398964 0.91606218 0.92020725 0.92331606 0.92849741 0.93264249
 0.93782383 0.94196891 0.94507772 0.95336788 0.95854922 0.96476684
 0.97202073 0.97720207 0.97927461 0.98238342 0.98756477 0.99170984
 0.99274611 0.99585492 1.        ]
True Positive Rates
[0.    0.002 0.006 0.006 0.012 0.02  0.026 0.028 0.03  0.034 0.04  0.042
 0.042 0.05  0.052 0.056 0.06  0.072 0.074 0.078 0.086 0.088 0.094 0.1
 0.102 0.116 0.122 0.132 0.14  0.146 0.15  0.152 0.16  0.166 0.168 0.178
 0.182 0.19  0.192 0.204 0.216 0.22  0.236 0.244 0.248 0.254 0.256 0.26
 0.266 0.27  0.284 0.29  0.298 0.314 0.318 0.32  0.322 0.332 0.34  0.346
 0.35  0.352 0.354 0.36  0.36  0.372 0.372 0.376 0.388 0.396 0.4   0.412
 0.416 0.422 0.424 0.436 0.44  0.44  0.442 0.446 0.448 0.454 0.458 0.466
 0.468 0.47  0.474 0.474 0.48  0.484 0.486 0.49  0.494 0.5   0.502 0.504
 0.504 0.508 0.512 0.514 0.514 0.518 0.522 0.53  0.532 0.536 0.542 0.544
 0.548 0.548 0.554 0.558 0.56  0.562 0.562 0.564 0.568 0.57  0.57  0.574
 0.574 0.576 0.578 0.58  0.58  0.588 0.594 0.598 0.602 0.606 0.608 0.61
 0.612 0.614 0.618 0.62  0.624 0.624 0.624 0.626 0.632 0.638 0.638 0.646
 0.646 0.65  0.65  0.654 0.654 0.658 0.666 0.666 0.67  0.67  0.672 0.676
 0.678 0.68  0.684 0.69  0.69  0.692 0.692 0.696 0.696 0.7   0.7   0.702
 0.702 0.706 0.708 0.712 0.712 0.72  0.72  0.722 0.722 0.724 0.726 0.734
 0.738 0.74  0.742 0.75  0.752 0.752 0.752 0.752 0.754 0.76  0.76  0.76
 0.766 0.772 0.776 0.778 0.78  0.78  0.788 0.788 0.788 0.792 0.792 0.794
 0.8   0.8   0.8   0.804 0.804 0.808 0.812 0.818 0.82  0.82  0.822 0.828
 0.828 0.828 0.828 0.828 0.836 0.838 0.84  0.842 0.842 0.842 0.844 0.848
 0.848 0.85  0.86  0.86  0.864 0.864 0.868 0.87  0.874 0.874 0.876 0.88
 0.88  0.882 0.884 0.888 0.888 0.89  0.89  0.892 0.894 0.898 0.902 0.902
 0.904 0.906 0.906 0.906 0.91  0.91  0.914 0.914 0.914 0.914 0.914 0.916
 0.916 0.916 0.916 0.916 0.918 0.918 0.918 0.918 0.918 0.918 0.92  0.92
 0.922 0.922 0.922 0.922 0.922 0.922 0.922 0.922 0.924 0.924 0.924 0.924
 0.924 0.926 0.928 0.928 0.932 0.932 0.934 0.934 0.934 0.936 0.936 0.936
 0.936 0.936 0.936 0.936 0.936 0.938 0.938 0.938 0.938 0.938 0.938 0.938
 0.938 0.938 0.938 0.94  0.942 0.942 0.944 0.944 0.948 0.948 0.952 0.952
 0.952 0.952 0.952 0.952 0.952 0.958 0.958 0.958 0.962 0.962 0.962 0.962
 0.962 0.964 0.966 0.968 0.968 0.968 0.97  0.972 0.972 0.972 0.972 0.972
 0.974 0.974 0.976 0.978 0.978 0.978 0.978 0.986 0.986 0.986 0.986 0.986
 0.986 0.986 0.986 0.988 0.988 0.988 0.988 0.988 0.988 0.988 0.99  0.992
 0.994 0.994 0.994 0.994 0.994 0.994 0.994 0.994 0.994 0.994 0.994 0.994
 0.996 0.996 0.996 0.996 0.996 0.998 0.998 0.998 1.   ]
p_0 Thresholds
[0.95 0.87 0.87 0.87 0.86 0.86 0.86 0.86 0.85 0.85 0.85 0.84 0.84 0.84
```

```
 0.84 0.84 0.84 0.83 0.83 0.83 0.83 0.83 0.83 0.82 0.82 0.82 0.82 0.82
 0.81 0.81 0.81 0.81 0.81 0.81 0.81 0.8  0.8  0.8  0.8  0.8  0.8  0.79
 0.79 0.79 0.79 0.79 0.79 0.78 0.78 0.78 0.78 0.78 0.78 0.77 0.77 0.77
 0.77 0.77 0.77 0.77 0.76 0.76 0.76 0.76 0.76 0.76 0.75 0.75 0.75 0.75
 0.74 0.74 0.74 0.74 0.74 0.74 0.74 0.73 0.73 0.73 0.73 0.73 0.73 0.72
 0.72 0.72 0.72 0.71 0.71 0.71 0.71 0.71 0.71 0.7  0.7  0.7  0.7  0.69
 0.69 0.69 0.69 0.68 0.68 0.68 0.68 0.68 0.68 0.67 0.66 0.66 0.65 0.65
 0.64 0.64 0.61 0.6  0.59 0.59 0.57 0.49 0.48 0.47 0.46 0.46 0.45 0.45
 0.45 0.44 0.44 0.44 0.43 0.43 0.43 0.42 0.42 0.42 0.41 0.41 0.41 0.41
 0.41 0.4  0.4  0.39 0.39 0.39 0.39 0.39 0.39 0.38 0.38 0.38 0.37 0.37
 0.37 0.37 0.37 0.37 0.37 0.36 0.36 0.36 0.36 0.36 0.35 0.35 0.35 0.35
 0.34 0.34 0.34 0.34 0.34 0.34 0.34 0.34 0.33 0.33 0.33 0.33 0.33 0.33
 0.33 0.33 0.32 0.32 0.32 0.32 0.32 0.32 0.32 0.31 0.31 0.31 0.31 0.31
 0.31 0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.29 0.29 0.29 0.29 0.29 0.29
 0.29 0.28 0.28 0.28 0.28 0.28 0.28 0.28 0.27 0.27 0.27 0.27 0.27 0.27
 0.27 0.27 0.27 0.26 0.26 0.26 0.26 0.26 0.26 0.26 0.25 0.25 0.25 0.25
 0.25 0.25 0.24 0.24 0.24 0.24 0.24 0.24 0.23 0.23 0.23 0.23 0.22 0.22
 0.22 0.21 0.21 0.21 0.21 0.2  0.2  0.2  0.2  0.19 0.19 0.19 0.19 0.18
 0.18 0.18 0.18 0.14 0.14 0.14 0.14 0.13 0.13 0.13 0.13 0.12 0.12 0.12
 0.12 0.12 0.11 0.11 0.11 0.11 0.11 0.11 0.11 0.11 0.11 0.11 0.1  0.1
 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.09 0.09 0.09
 0.09 0.09 0.09 0.09 0.09 0.09 0.09 0.09 0.09 0.09 0.08 0.08 0.08 0.08
 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.07 0.07 0.07 0.07 0.07
 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.07 0.06 0.06
 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06 0.06
 0.06 0.06 0.06 0.06 0.06 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
 0.05 0.05 0.05 0.05 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04
 0.04]
```

What index(s) in the fprs list is equal to around 0.42?

In [36]:
```
np.where((fprs>.40) & (fprs<.44))
```

Out[36]:
```
(array([240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250], dtype=int6
4),)
```

What is the corresponding true positive rate and p_0 threshold (for a false positive rate of 0.4)?

In [37]:
```
print('True positive rate')
print(tprs[27])
print('p_0 threshold')
print(p_thresholds[27])
```

```
True positive rate
0.132
p_0 threshold
0.8162119939653578
```

Thus, using this threshold, how would our model classify a 20-year old, female, Democrat?

```
In [38]: mod2.predict(exog=dict(age=20, sex='Female', party='Democrat'))

Out[38]: 0    0.039545
         dtype: float64
```

STAT 207, Douglas Simpson, University of Illinois at Urbana-Champaign