Limited Dependent Variables

Today's Outline

- Limited dependent variables
    - Linear Probability Model
    - Logit
    - Probit
    - Interpretation
    - Prediction
    - Multinomial Logit
    - Poisson

Binary prediction Example

- The goal is to predict whether or not an individual chooses a car for transportation given some difference in the bus and car commutes (dtime)
- In other words we want to estimate:

$$P(Y = 1 | X) = P(Auto = 1 | dtime)$$

- We want the estimated probability to be between 0 and 1

```python
# replace data
data = pd.read_csv("transport.csv")
```

```python
data.head()
```

[21]:

	autotime	bustime	dtime	auto
0	52.9	4.4	-4.85	0
1	4.1	28.5	2.44	0
2	4.1	86.9	8.28	1
3	56.2	31.6	-2.46	0
4	51.8	20.2	-3.16	0

Linear Probability Model

- It is possible to run a typical linear regression to estimate the desired probability

$$P(Auto = 1 \mid dtime) = \beta_0 + \beta_1 dtime + e$$

- Why would this not always yield a desirable result?

```
ols1 = smf.ols('auto ~ dtime', data).fit()
ols1.summary()
```

OLS Regression Results

Dep. Variable:	auto	**R-squared:**	0.611
Model:	OLS	**Adj. R-squared:**	0.591
Method:	Least Squares	**F-statistic:**	29.88
Date:	Wed, 30 Nov 2022	**Prob (F-statistic):**	2.83e-05
Time:	15:42:49	**Log-Likelihood:**	-5.2951
No. Observations:	21	**AIC:**	14.59
Df Residuals:	19	**BIC:**	16.68
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.4848	0.071	6.785	0.000	0.335	0.634
dtime	0.0703	0.013	5.467	0.000	0.043	0.097

Omnibus:	2.283	**Durbin-Watson:**	1.979
Prob(Omnibus):	0.319	**Jarque-Bera (JB):**	0.807
Skew:	0.293	**Prob(JB):**	0.668
Kurtosis:	3.761	**Cond. No.**	5.56

```
ols1.fittedvalues.values
```

```
]: array([ 0.14379198,  0.65635127,  1.06696118,  0.31183268,  0.26261573,
           1.12461531,  0.85110974, -0.1318229 ,  0.36526821,  0.122699  ,
          -0.15291587,  0.94532502,  0.17543144,  0.43557813,  0.84759424,
           0.7125992 ,  0.05027979,  0.72384879,  0.68095974, -0.02776422,
           0.83564155])
```

```
# we can still assign them a value based on some cutoff
np.where(ols1.fittedvalues >.5, 1, 0)
```

```
]: array([0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1])
```

Logit Model

- The logit model can be broken into the link function G() and the linear function inside the link function

$$P(y=1|x) = G(\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k)$$

- The smf.logit() performs maximum likelihood estimation of the betas automatically
- The link function takes in the values of the linear model and always outputs a value between 0 and 1
- The link function for the probit model is the standard normal CDF:

$$G(z) = \frac{exp(z)}{1+exp(z)}$$

- The function also returns the pseudo r-squared and tests of significance

```
logit = smf.logit('auto ~ dtime', data).fit()
logit.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.293621
         Iterations 7
```

]:

Logit Regression Results

Dep. Variable:	auto	**No. Observations:**	21
Model:	Logit	**Df Residuals:**	19
Method:	MLE	**Df Model:**	1
Date:	Wed, 30 Nov 2022	**Pseudo R-squ.:**	0.5757
Time:	15:43:18	**Log-Likelihood:**	-6.1660
converged:	True	**LL-Null:**	-14.532
Covariance Type:	nonrobust	**LLR p-value:**	4.304e-05

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.2376	0.750	-0.317	0.752	-1.708	1.233
dtime	0.5311	0.206	2.573	0.010	0.127	0.936

nterpretation of the coefficients

Probit Model

- The probit model can be broken into the link function G() and the linear function inside the link function

$$P(y=1|x) = G(\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k)$$

- The smf.probit() performs maximum likelihood estimation of the betas automatically
- The link function takes in the values of the linear model and always outputs a value between 0 and 1
- The link function for the probit model is the standard normal CDF:

$$G(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(z)^2}$$

- The function also returns the pseudo r-squared and tests of significance

```
probit1 = smf.probit('auto ~ dtime', data).fit(disp = 0)
probit1.summary()
```

Probit Regression Results

Dep. Variable:	auto	**No. Observations:**	21
Model:	Probit	**Df Residuals:**	19
Method:	MLE	**Df Model:**	1
Date:	Wed, 30 Nov 2022	**Pseudo R-squ.:**	0.5758
Time:	15:45:16	**Log-Likelihood:**	-6.1652
converged:	True	**LL-Null:**	-14.532
Covariance Type:	nonrobust	**LLR p-value:**	4.300e-05

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0644	0.399	-0.161	0.872	-0.847	0.718
dtime	0.3000	0.103	2.916	0.004	0.098	0.502

Coefficient Interpretations (Partial Effects)

- The approximate change in the probability y = 1 for a one-unit increase in x can be found at a particular point by applying:

$$\hat{\beta}_j * g(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_k x_k)$$

- Where g() is the pdf of the link function

```
beta_j = logit.params[1]
linear = logit.fittedvalues[0]
partial= logit.params[1]*stats.logistic.pdf(linear)

# the partial effect of x_j on y changes depending on the values of the other regressors
print("The approximate change in the probability that someone will commute in a car at " +
      str(linear), "is " + str(partial))
```

```
The approximate change in the probability that someone will commute in a car at -2.8134020769391905 is 0.02836075171600828
```

```
beta_j = logit.params[1]
linear = logit.fittedvalues[2]
partial= logit.params[1]*stats.logistic.pdf(linear)

# Note that the increase in probability is much smaller
print("The approximate change in the probability that someone will commute in a car at " +
      str(linear), "is " + str(partial))
```

```
The approximate change in the probability that someone will commute in a car at 4.1599182693181325 is 0.008036971009232381
```

Partial Effects at the Average (PEA)

- One quick way to get a quick measure of the partial effect of a variable is take the average of the x variables and apply the partial effect formula

$$PEA = \hat{\beta}_j * g(\bar{x}\hat{\beta})$$

```
logit_pea = logit.params*stats.logistic.pdf(logit.params[0] + logit.params[1]*data.dtime.mean())
probit_pea = probit1.params*stats.norm.pdf(probit1.params[0] + probit1.params[1]*data.dtime.mean())

print("Logistic APE: ", logit_ape, "Probit PEA: ", probit_pea)
```

```
Logistic APE:  Intercept   -0.020646
dtime        0.046154
dtype: float64 Logistic PEA:  Intercept   -0.058055
dtime        0.129781
dtype: float64
```

Average Partial Effects

- The PEA method is often not preferred since averages don't make sense for binary variables (male/female) and transformed continuous variables
- The APE method takes the average of the pdf over the fitted values

$$APE = \hat{\beta}_j * \overline{g(x\hat{\beta})}$$

```
probit_ape = probit1.params*stats.norm.pdf(probit1.fittedvalues).mean()
logit_ape = logit.params*stats.logistic.pdf(logit.fittedvalues).mean()

print("Probit APE: ", probit_ape, "Probit PEA: ", probit_pea)
```

```
Probit APE:  Intercept   -0.010397
dtime        0.048407
dtype: float64 Probit PEA:  Intercept   -0.025574
dtime        0.119068
dtype: float64
```

Average Partial Effects (Method)

- The get_margeff() method will automatically calculate the APE for us

Marginal Effects (Automatically calculate the APE)

```
probit1.get_margeff().margeff
```

```
89]: array([0.04840694])
```

```
logit.get_margeff().margeff
```

```
90]: array([0.04615397])
```

Predictions

$$P(y=1|x) = G(\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k)$$

There are three separate prediction methods to be aware of for fitted logit and probit models:

- results.fittedvalues gives:

$$x_i\hat{\beta}$$

```
# below we have the fitetd values from the logit model
# note these are not probabilities
print(logit.fittedvalues.min())
print(logit.fittedvalues.max())
```

```
-1.2980561792904026
0.6952090574421678
```

- results.predict() gives:

$$G(x_i\hat{\beta})$$

```
logit.predict()
```

```
array([0.05660423, 0.74236637, 0.98463106, 0.17594335, 0.12832551,
       0.99000296, 0.92618052, 0.00742605, 0.24223912, 0.04867308,
       0.00633924, 0.96235264, 0.07080376, 0.3522088 , 0.92434429,
       0.81505287, 0.02875512, 0.82752096, 0.77629228, 0.01615431,
       0.91778346])
```

- results.predict(newdata) can tell us what our model says for unseen value s

```
# create some new values to predict with the same
# regressor names
newdata = pd.DataFrame([1,2,3,4], columns = ["dtime"])
```

```
logit.predict(newdata)
```

```
0    0.572858
1    0.695216
2    0.795063
3    0.868392
dtype: float64
```

Predictions (Probit)

- Predictions are made the same way with the probit model
- Ultimately we need to decide whether a prediction is 0 or 1
- On the right the threshold is set to .5

```
import scipy.stats as stats

# convert to probabilities
stats.norm.cdf(probit1.fittedvalues)[:10]
```

```
array([0.61478336, 0.57195901, 0.60921791, 0.6348224 , 0.56892911,
       0.61993933, 0.62339582, 0.61483263, 0.58300447, 0.60484237])
```

```
# get the probabilities for teh fitted values
probit1.predict()[:10]
```

```
array([0.61478336, 0.57195901, 0.60921791, 0.6348224 , 0.56892911,
       0.61993933, 0.62339582, 0.61483263, 0.58300447, 0.60484237])
```

```
# predict new values
probit1.predict(newdata)
```

```
0    0.357771
1    0.703864
2    0.798292
3    0.871922
dtype: float64
```

```
# Set a threshold where we will predict 1 if the probability is greater than .5
np.where(probit1.predict(newdata) >.5, 1, 0)
```

```
array([0, 1, 1, 1])
```

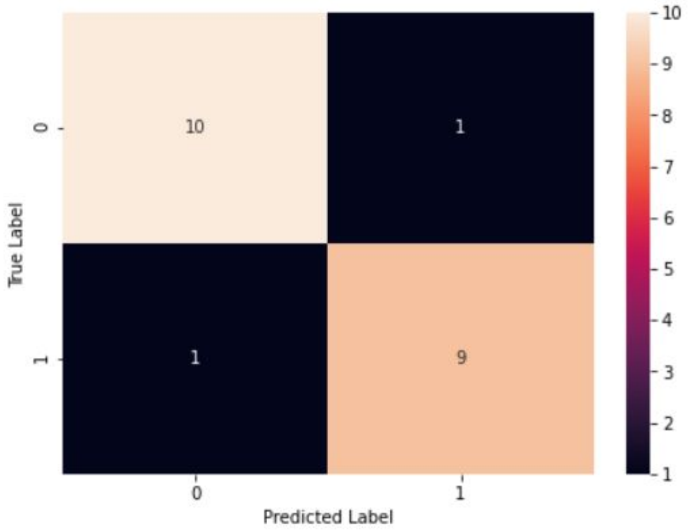
Confusion Matrix

- Looking at accuracy (percent predicted correctly) alone can be misleading
- For example, if negatives are rare, a high accuracy can be achieved by only ever guessing positive
- We can visualize the relationships between true or false positives in a confusion matrix

```python
from sklearn.metrics import confusion_matrix
predictions = np.where(logit.predict() > .5, 1, 0)
cm = confusion_matrix(actual, predictions)\
actual = data.auto
cm
```

```
array([[10,  1],
       [ 1,  9]], dtype=int64)
```

```python
import seaborn as sn
plt.figure(figsize = (7,5))
sn.heatmap(cm, annot=True)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Multinomial Logistic Regression

- The mnlogit() function gives the coefficient estimates for each level of the response variable
- Below we have reproduced an example for school choice from ECON 430
- The response variable can take three values
- We can also get the marginal effect of each variable for each level

```
# each colyumn represents the prbability y takes a given value
results_mn.predict()[:3]
```

```
array([[0.42276441, 0.35590784, 0.22132775],
       [0.33928751, 0.3583217 , 0.30239078],
       [0.24869707, 0.34134015, 0.40996277]])
```

```
results_mn.predict().argmax(axis = 1)[:10]
```

```
array([0, 1, 2, 2, 2, 2, 0, 0, 0, 2], dtype=int64)
```

```
results_mn = smf.mnlogit('psechoice ~ grades', nels).fit(disp = 0)
```

```
results_mn.summary()
```

MNLogit Regression Results

Dep. Variable:	psechoice	**No. Observations:**	6649
Model:	MNLogit	**Df Residuals:**	6645
Method:	MLE	**Df Model:**	2
Date:	Wed, 30 Nov 2022	**Pseudo R-squ.:**	0.1360
Time:	16:35:10	**Log-Likelihood:**	-5869.6
converged:	True	**LL-Null:**	-6793.2
Covariance Type:	nonrobust	**LLR p-value:**	0.000

psechoice=2	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	2.5015	0.157	15.955	0.000	2.194	2.809
grades	-0.2945	0.020	-14.938	0.000	-0.333	-0.256
psechoice=3	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	5.6268	0.153	36.794	0.000	5.327	5.927
grades	-0.6910	0.020	-34.141	0.000	-0.731	-0.651

```
me = results_mn.get_margeff(at =  'overall', method = 'dydx')
```

```
# get the APE of the predictors for each level
# hypothesis test are automatically computed
me.summary_frame()
```

endog	**exog**	**dy/dx**	**Std. Err.**	**z**	**Pr(>|z|)**	**Conf. Int. Low**	**Cont. Int. Hi.**
psechoice=1	**grades**	0.069040	0.001969	35.064846	2.316117e-269	0.065181	0.072899
psechoice=2	**grades**	0.028805	0.002102	13.702720	9.779134e-43	0.024685	0.032925
psechoice=3	**grades**	-0.097845	0.001671	-58.537530	0.000000e+00	-0.101121	-0.094569

Multinomial Logistic Regression

- The mnlogit() function gives the coefficient estimates for each level of the response variable
- Below we have reproduced an example for school choice from ECON 430
- The response variable can take three values
- We can also get the marginal effect of each variable for each level

```
# each colyumn represents the prbability y takes a given value
results_mn.predict()[:3]
```

```
]: array([[0.42276441, 0.35590784, 0.22132775],
          [0.33928751, 0.3583217 , 0.30239078],
          [0.24869707, 0.34134015, 0.40996277]])
```

```
results_mn.predict().argmax(axis = 1)[:10]
```

```
]: array([0, 1, 2, 2, 2, 2, 0, 0, 0, 2], dtype=int64)
```

```
results_mn = smf.mnlogit('psechoice ~ grades', nels).fit(disp = 0)
```

```
results_mn.summary()
```

]:

MNLogit Regression Results

Dep. Variable:	psechoice	**No. Observations:**	6649
Model:	MNLogit	**Df Residuals:**	6645
Method:	MLE	**Df Model:**	2
Date:	Wed, 30 Nov 2022	**Pseudo R-squ.:**	0.1360
Time:	16:35:10	**Log-Likelihood:**	-5869.6
converged:	True	**LL-Null:**	-6793.2
Covariance Type:	nonrobust	**LLR p-value:**	0.000

psechoice=2	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	2.5015	0.157	15.955	0.000	2.194	2.809
grades	-0.2945	0.020	-14.938	0.000	-0.333	-0.256
psechoice=3	**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	5.6268	0.153	36.794	0.000	5.327	5.927
grades	-0.6910	0.020	-34.141	0.000	-0.731	-0.651

```
me = results_mn.get_margeff(at =  'overall', method = 'dydx')
```

```
# get the APE of the predictors for each level
# hypothesis test are automatically computed
me.summary_frame()
```

endog	exog	dy/dx	Std. Err.	z	Pr(>|z|)	Conf. Int. Low	Cont. Int. Hi.
psechoice=1	**grades**	0.069040	0.001969	35.064846	2.316117e-269	0.065181	0.072899
psechoice=2	**grades**	0.028805	0.002102	13.702720	9.779134e-43	0.024685	0.032925
psechoice=3	**grades**	-0.097845	0.001671	-58.537530	0.000000e+00	-0.101121	-0.094569

Classification Exercise

Use the LOANAPP data from

- Estimate a probit model of approve on white. Find the estimated probability of loan approval for both whites and nonwhites. How do these compare with the linear probability model?
- Add the variables unem, male, and married to the model and reestimate.
- What is the APE for each predictor? How do they compare?
- What is the probability of being a married, non-white, man, with a net worth of 1,000 being approved?

Poisson Regression

- A poisson regression is used when we have data where the dependent variable is some sort of count, y = 0, 1, 2, 3…
- Examples could be trips to the doctor, arrests, cigarettes smoked, etc.
- For a poisson random variable, we know:

$$f(y) = \Pr(Y = y) = \frac{e^{-\lambda}\lambda^{y}}{y!}, \quad y = 0, 1, 2, \ldots$$

- Prediction of the conditional mean of y for a given observation is just:

$$\widehat{E}(y_0) = \widehat{\lambda}_0 = \exp\left(\widehat{\beta}_1 + \widehat{\beta}_2 x_0\right)$$

Poisson Regression

- A poisson regression can be performed in statsmodels
- The betas in the summary correspond are used to make a linear prediction
- These linear predictions are used to make the mean prediction (of cigarette consumption for example)

$$\widehat{E}\left(y_0\right) = \widehat{\lambda}_0 = \exp\left(\widehat{\beta}_1 + \widehat{\beta}_2 x_0\right)$$

```python
# estimate Poisson model:
reg_poisson = smf.poisson(formula='cigs ~ cigprice + cigtax + lfaminc',
                          data=birth)
results_poisson = reg_poisson.fit(disp=0)
```

```python
results_poisson.summary()
```

Poisson Regression Results

Dep. Variable:	cigs	**No. Observations:**	1388
Model:	Poisson	**Df Residuals:**	1384
Method:	MLE	**Df Model:**	3
Date:	Thu, 16 Nov 2023	**Pseudo R-squ.:**	0.04280
Time:	14:13:17	**Log-Likelihood:**	-6166.2
converged:	True	**LL-Null:**	-6441.9
Covariance Type:	nonrobust	**LLR p-value:**	3.517e-119

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	2.2532	0.417	5.410	0.000	1.437	3.069
cigprice	-0.0062	0.004	-1.580	0.114	-0.014	0.001
cigtax	0.0196	0.005	3.811	0.000	0.010	0.030
lfaminc	-0.3831	0.016	-24.452	0.000	-0.414	-0.352

Prediction Using a Poisson Regression

- Statsmodels provides options for the type of prediction you would like to make

$$\widehat{E}\left(y_0\right) = \widehat{\lambda}_0 = \boxed{\exp\left(\boxed{\widehat{\beta}_1 + \widehat{\beta}_2 x_0}\right)}$$

which : 'mean', 'linear', 'var', 'prob' (`optional`)

Statitistic to predict. Default is 'mean'.

- 'mean' returns the conditional expectation of endog E(y | x), i.e. exp of linear predictor.
- 'linear' returns the linear predictor of the mean function.
- 'var' returns the estimated variance of endog implied by the model.
- 'prob' return probabilities for counts from 0 to max(endog) or for y_values if those are provided.

$$\widehat{\Pr}\left(Y = y\right) = \frac{\exp\left(\widehat{\lambda}\right)\widehat{\lambda}^{y}}{y!},$$

*Note that to get the probability you have to divide the number above by the sum of total probabilities for a given lambda

Prediction Using a Poisson Regression (Linear and Mean)

$$\widehat{E}(y_0) = \widehat{\lambda}_0 = \boxed{\exp\left(\boxed{\widehat{\beta}_1 + \widehat{\beta}_2 x_0}\right)}$$

```python
# the linear prediction is just making a prediction
# as you would any regression
np.array(results_poisson.params)@X.T
```

```
0       0.825926
1       1.051128
2       2.088676
3       0.772996
4       0.553327
          ...
1383    0.719256
1384    1.335888
1385    0.080019
1386    0.409592
1387    0.290761
Length: 1388, dtype: float64
```

```python
results_poisson.predict(birth, which = 'linear')
```

```
0       0.825926
1       1.051128
2       2.088676
3       0.772996
4       0.553327
          ...
1383    0.719256
1384    1.335888
1385    0.080019
1386    0.409592
1387    0.290761
```

```python
np.exp(np.array(results_poisson.params)@X.T)
```

```
0       2.283995
1       2.860875
2       8.074217
3       2.166246
4       1.739029
          ...
1383    2.052906
1384    3.803373
1385    1.083307
1386    1.506203
1387    1.337445
Length: 1388, dtype: float64
```

```python
results_poisson.predict(birth, which = 'mean')
```

```
0       2.283995
1       2.860875
2       8.074217
3       2.166246
4       1.739029
          ...
1383    2.052906
1384    3.803373
1385    1.083307
1386    1.506203
1387    1.337445
Length: 1388, dtype: float64
```

Prediction Using a Poisson Regression (Probability)

- Which = ‘prob’ will give you the probability Y = y for each observation over which you make a prediction

```
probs0 = results_poisson.predict(birth, which = 'prob')
```

```
probs0
```

	0	1	2	3	4	5	6	7	8	9	...	41	42	43	4
0	0.101876	0.232685	0.265726	0.202306	0.115516	0.052768	0.020087	0.006554	0.001871	0.000475	...	1.549336e-36	8.425416e-38	4.475257e-39	2.323060e 4
1	0.057219	0.163695	0.234156	0.223297	0.159706	0.091380	0.043571	0.017807	0.006368	0.002024	...	8.903577e-33	6.064768e-34	4.035010e-35	2.623559e 3
2	0.000311	0.002515	0.010153	0.027325	0.055157	0.089070	0.119862	0.138256	0.139539	0.125185	...	1.445773e-16	2.779401e-17	5.218950e-18	9.577031e 1
3	0.114607	0.248267	0.268904	0.194171	0.105155	0.045558	0.016448	0.005090	0.001378	0.000332	...	1.989779e-37	1.026274e-38	5.170146e-40	2.545411e 4
4	0.175691	0.305532	0.265664	0.153999	0.066952	0.023286	0.006749	0.001677	0.000364	0.000070	...	3.740271e-41	1.548676e-42	6.263240e-44	2.475445e 4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
1383	0.128361	0.263514	0.270484	0.185093	0.094995	0.039003	0.013345	0.003914	0.001004	0.000229	...	2.461154e-38	1.202981e-39	5.743271e-41	2.679636e 4
1384	0.022295	0.084798	0.161259	0.204443	0.194393	0.147870	0.093734	0.050929	0.024213	0.010232	...	4.080476e-28	3.695136e-29	3.268367e-30	2.825186e 3
1385	0.338474	0.366672	0.198609	0.071718	0.019423	0.004208	0.000760	0.000118	0.000016	0.000002	...	2.691029e-49	6.940980e-51	1.748655e-52	4.305297e 5

Prediction Using a Poisson Regression (Probability)

- The dataframe of probabilities in the previous slide is constructed using teh following procedure for each Y

```
probs = np.array([])

# for each possible count
for y in range(0,51):
    # calculate the mean
    l = results_poisson.predict(birth, which = 'mean')

    # plug the estimated lambda into the pdf for the poisson distribution
    probs = np.append(probs,((l[0]**y)*np.exp(l[0]))/np.math.factorial(y))
```

```
# Take the sum and divide the estimated probabilities by 1 to get the
# probability that the observation is equal to a given count
ps = pd.DataFrame(probs/probs.sum())
```

```
ps.round(4)[:12]
```

	0
0	0.1019
1	0.2327
2	0.2657
3	0.2023
4	0.1155
5	0.0528
6	0.0201
7	0.0066
8	0.0019
9	0.0005
10	0.0001
11	0.0000

```
probs0.loc[0].round(4)[:12]

0      0.1019
1      0.2327
2      0.2657
3      0.2023
4      0.1155
5      0.0528
6      0.0201
7      0.0066
8      0.0019
9      0.0005
10     0.0001
11     0.0000
Name: 0, dtype: float64
```

$$\widehat{\Pr}(Y = y) = \frac{\exp\left(\hat{\lambda}\right)\hat{\lambda}^y}{y!},$$