

Regression

Assoc. Prof. Peerapon Vateekul, Ph.D.

Peerapon.v@chula.ac.th

CHULA ENGINEERING
Foundation toward Innovation

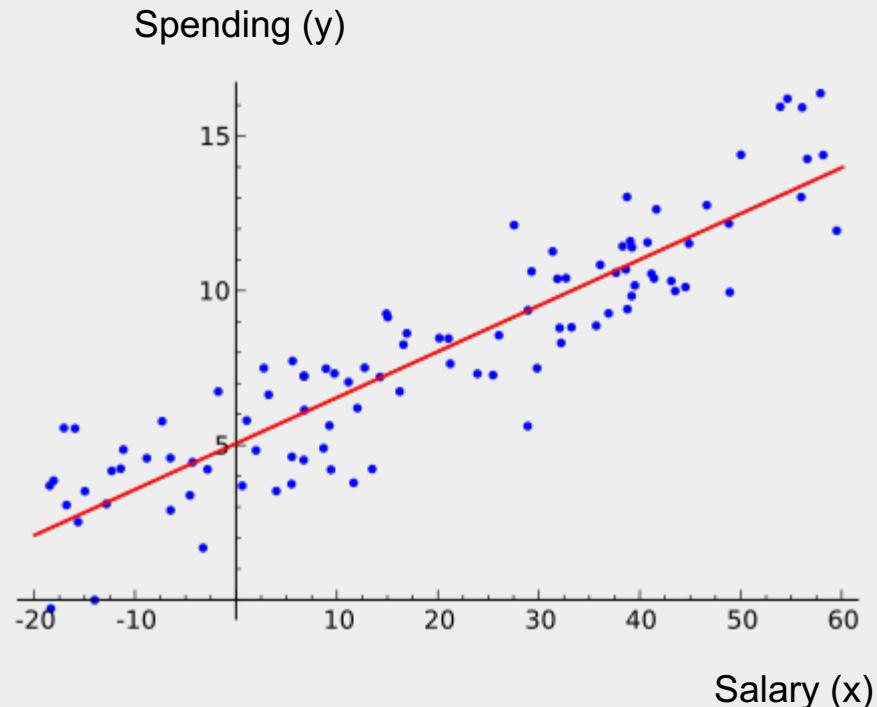


Chula Big Data and IoT
Center of Excellence (CUBIC)

Linear Regression

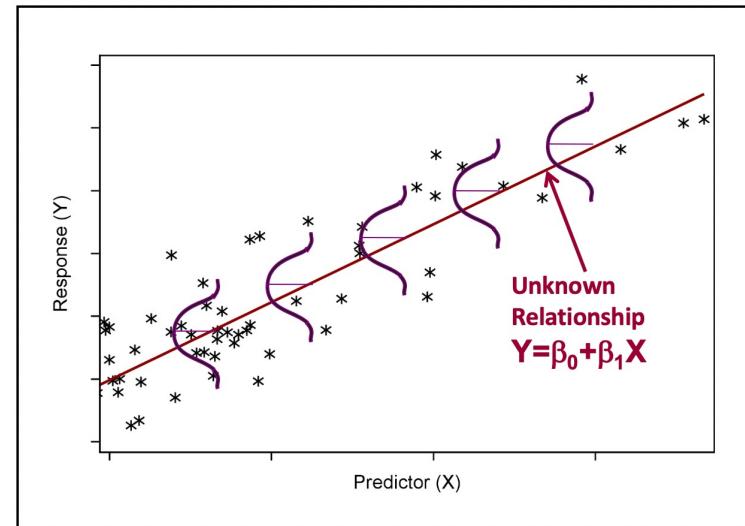
Build a supervised learning model using Excel

Type of Response \ Type of Predictors	Categorical	Continuous	Continuous and Categorical
Continuous	Analysis of Variance (ANOVA)	Ordinary Least Squares (OLS) Regression	Analysis of Covariance (ANCOVA)
Categorical	Contingency Table Analysis or Logistic Regression	Logistic Regression	Logistic Regression



Linear Regression: Model

- Main Idea: **to obtain a line that best fits the data** where its overall prediction error from all data points are as small as possible.
- Simple Linear Regression (1 feature):
 - $y(x) = \theta_0 + \theta_1 x_1$
- Multiple Linear Regression (n features):
 - $y(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
- 4 key assumptions of linear regression:
 - Linear relationship
 - Multivariate normality
 - No or little multicollinearity (no auto-correlation)
 - Homoscedasticity





Simple Linear Regression



Problem: 1 input (predictor) & 1 output

- Collect data of 7 patients
- Systolic Blood Pressure (y) & Cholesterol (x)

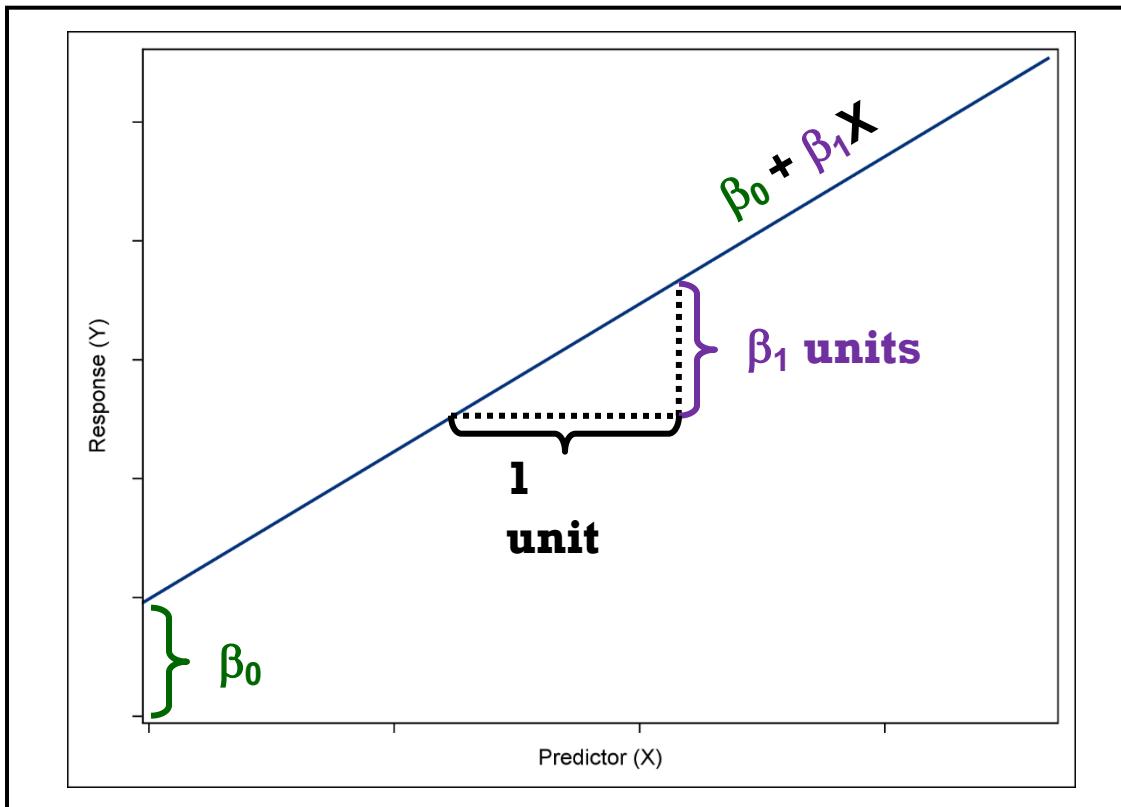
idno	chol (x)	sysbp (y)
1	437	194
2	264	121
3	249	131
4	297	159
5	243	123
6	272	161
7	161	115
总数	1923	1004



$$\hat{y} = \beta_0 + \beta_1 x$$

$$\widehat{bp} = \beta_0 + \beta_1 chol$$

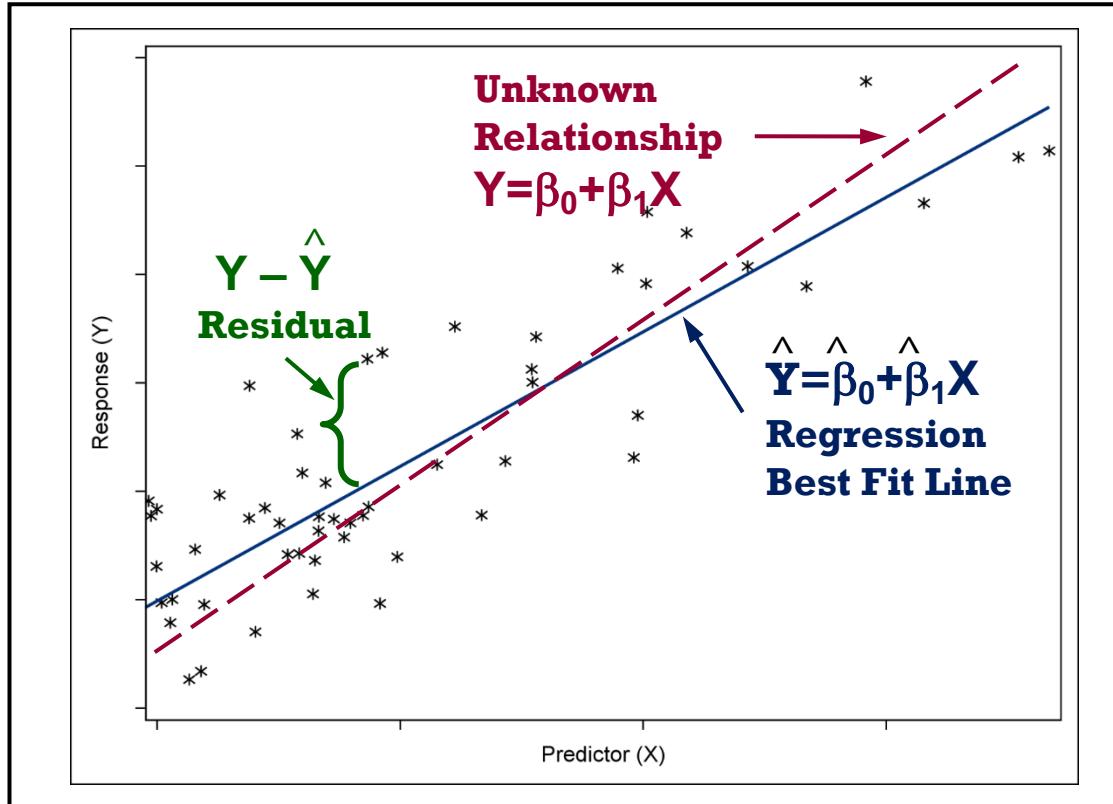
Simple Linear Regression Model



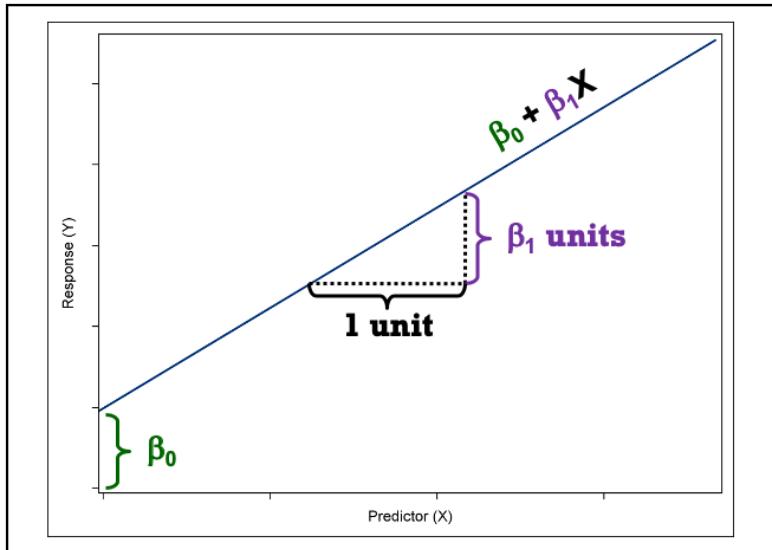
$$\hat{y} = \beta_0 + \beta_1 x$$

$$\widehat{bp} = \beta_0 + \beta_1 chol$$

Ordinary Least Squares (OLS) Regression



How to estimate parameters



$$\hat{y} = \beta_0 + \beta_1 x$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

$$\beta_1 = \frac{s_{xy}}{s_{xx}} = \frac{\left(\sum x_i y_i - \frac{\sum x_i \sum y_i}{n} \right)}{\left(\sum x_i^2 - \frac{(\sum x_i)^2}{n} \right)}$$

$$[Y] = [X][\beta]$$

$$[\beta] = [X]^{-1}[Y]$$

Example

$$\hat{y} = \beta_0 + \beta_1 x$$

- Systolic Blood Pressure (y)
- Cholesterol (x)

idno	chol (x)	sysbp (y)	x²	xy	y²
1	437	194	190969	84778	37636
2	264	121	69696	31944	14641
3	249	131	62001	32619	17161
4	297	159	88209	47223	25281
5	243	123	49049	29889	15129
6	272	161	73984	43792	25921
7	161	115	25921	18515	13225
รวม	1923	1004	569829	288760	148994

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

$$\beta_1 = \frac{s_{xy}}{s_{xx}} = \frac{\left(\sum x_i y_i - \frac{\sum x_i \sum y_i}{n} \right)}{\left(\sum x_i^2 - \frac{(\sum x_i)^2}{n} \right)}$$

$$\bar{x} = \frac{1923}{7} = 247.7143, \bar{y} = \frac{1004}{7} = 143.4286$$

$$\beta_1 = \frac{s_{xy}}{s_{xx}} = \frac{\left(288760 - \frac{1923 \times 1004}{7} \right)}{\left(569829 - \frac{(1923)^2}{7} \right)} = 0.3116$$

$$\beta_0 = 143.4286 - (0.3116)(247.7143) = 57.8355$$

$$\hat{y} = 57.8355 + 0.3116x$$

$$\widehat{bp} = 57.8355 + 0.3116 \times chol$$

How to read an equation

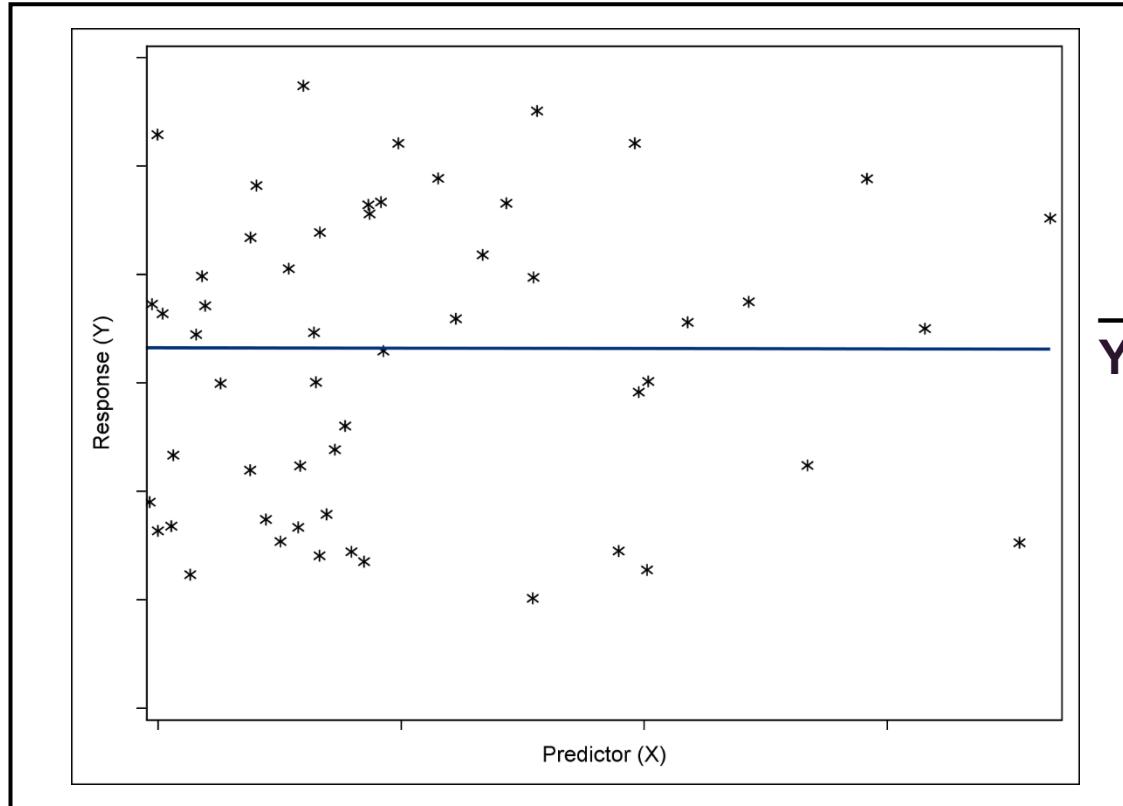
Example: Prediction

- Systolic Blood Pressure (y)
- Cholesterol (x)

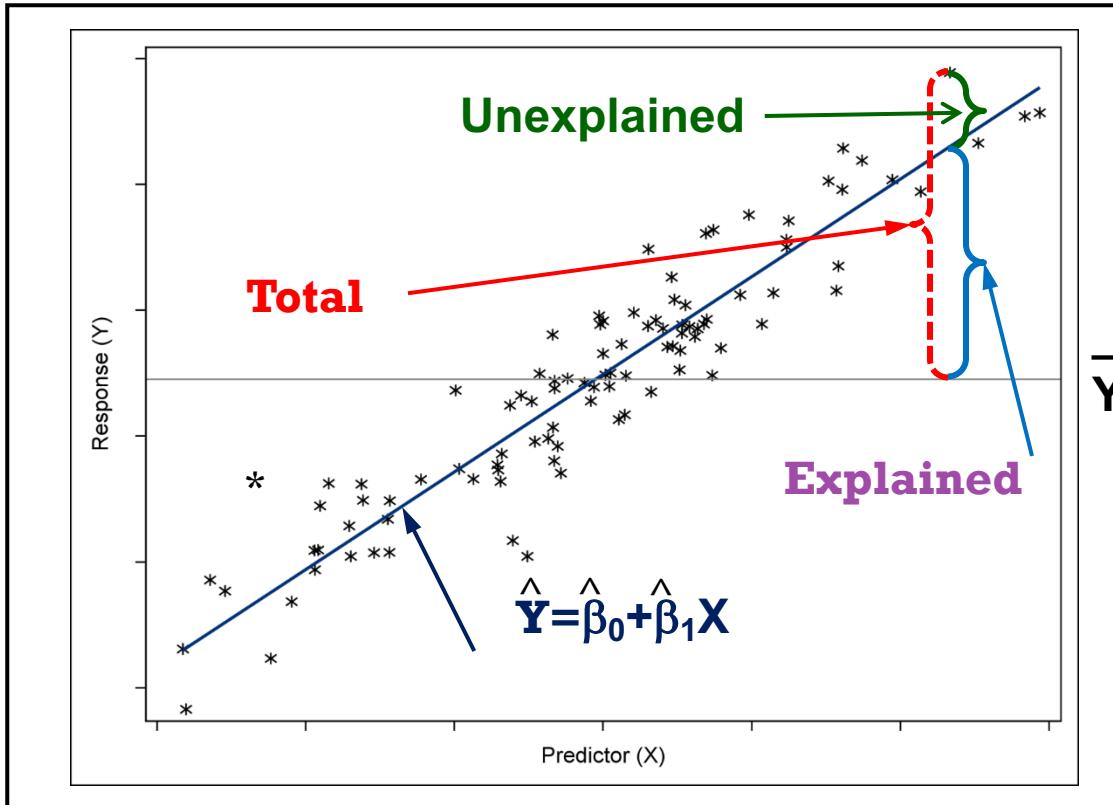
$$\widehat{bp} = 57.8355 + 0.3116 \times chol$$

idno	chol(x)	sysbp(y)	predict
1	437	194	196.1897
2	264	121	141.4179
3	249	131	136.6689
4	297	159	151.8657
5	243	123	134.7693
6	272	161	143.9507
7	161	115	108.8081
MEAN	1923	1004	

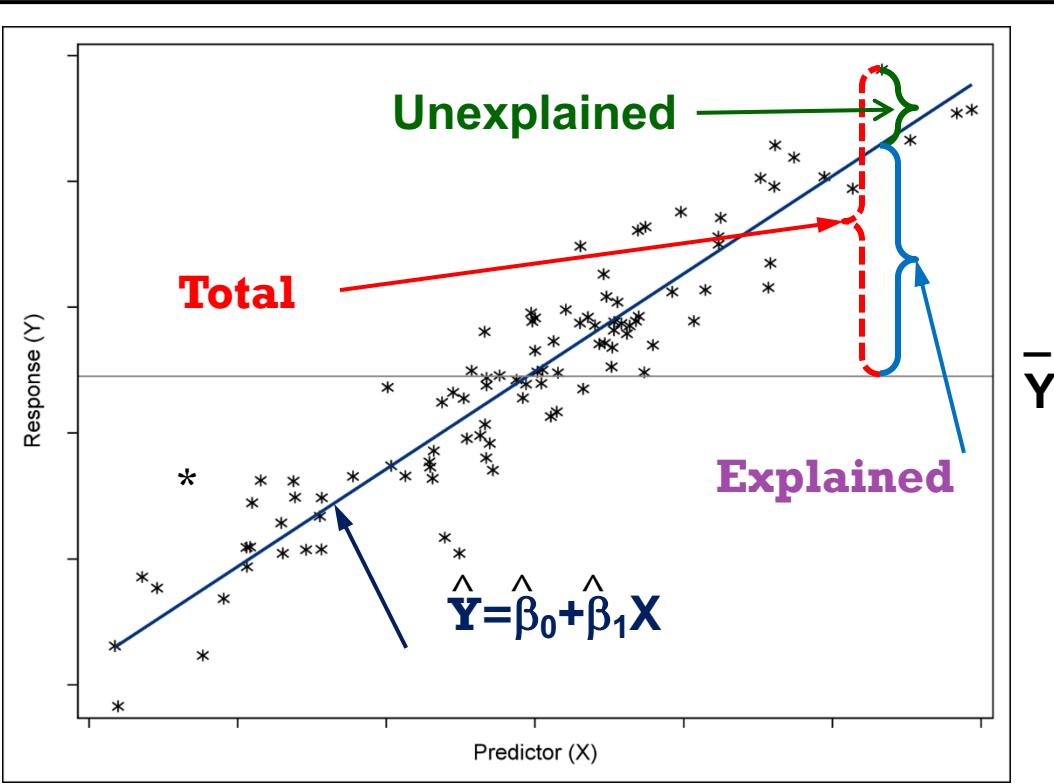
The Baseline Model (Null Hypothesis)



Explained versus Unexplained Variability



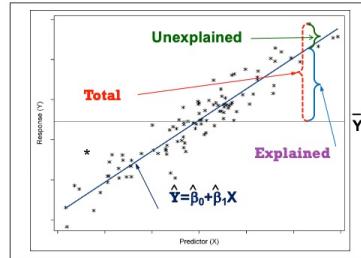
Coefficient of Determination



$$R^2 = \frac{SSM}{SST} = 1 - \frac{SSE}{SST}$$

- “Proportion of variance accounted for by the model”

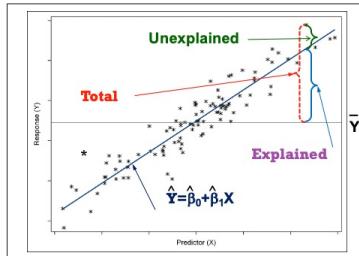
Coefficient of Determination (cont.)



$$R^2 = \frac{SSM}{SST} = 1 - \frac{SSE}{SST}$$

id	chol (x)	bp (y)	predict	error	squared error (SE)	guess	(y - y_bar)	squared total (ST)
1	437	194	196.1897	(2.1897)	4.7948	143.4286	50.5714	2,557.4694
2	264	121	141.4179	(20.4179)	416.8906	143.4286	(22.4286)	503.0408
3	249	131	136.6689	(5.6689)	32.1364	143.4286	(12.4286)	154.4694
4	297	159	151.8657	7.1343	50.8982	143.4286	15.5714	242.4694
5	243	123	134.7693	(11.7693)	138.5164	143.4286	(20.4286)	417.3265
6	272	161	143.9507	17.0493	290.6786	143.4286	17.5714	308.7551
7	161	115	108.8081	6.1919	38.3396	143.4286	(28.4286)	808.1837
average	274.7143	143.4286		SSE	972.2548		SST	4,991.7143
)				MSE	138.8935			
-				RMSE	11.7853			
:								
:	R^2	1 - (SSE/SST)	0.8052					

Coefficient of Determination (cont.)



- Train: R^2 , RMSE
- Test: R^2 , RMSE (honest estimate)

Training Data



Testing Data

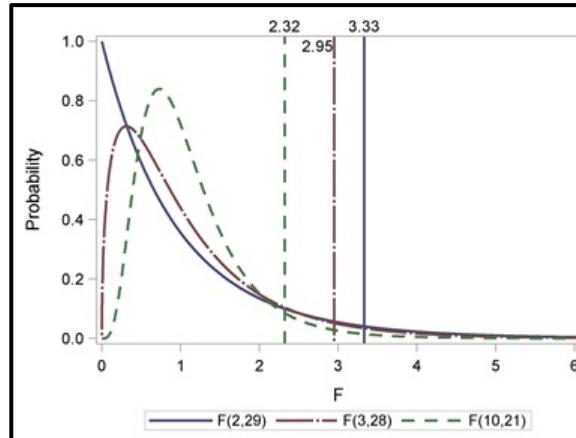


id	chol (x)	bp (y)	predict	error	squared error (SE)	guess	(y - y_bar)	squared total (ST)
1	437	194	196.1897	(2.1897)	4.7948	143.4286	50.5714	2,557.4694
2	264	121	141.4179	(20.4179)	416.8906	143.4286	(22.4286)	503.0408
3	249	131	136.6689	(5.6689)	32.1364	143.4286	(12.4286)	154.4694
4	297	159	151.8657	7.1343	50.8982	143.4286	15.5714	242.4694
5	243	123	134.7693	(11.7693)	138.5164	143.4286	(20.4286)	417.3265
6	272	161	143.9507	17.0493	290.6786	143.4286	17.5714	308.7551
7	161	115	108.8081	6.1919	38.3396	143.4286	(28.4286)	808.1837
average	274.7143	143.4286		SSE	972.2548		SST	4,991.7143
)				MSE	138.8935			
-				RMSE	11.7853			
;	R^2	1 - (SSE/SST)	0.8052					

Model Hypothesis Test

F Statistic and Critical Values at $\alpha=0.05$

- Null Hypothesis:
 - The simple linear regression model does not fit the data better than the baseline model.
 - $\beta_1=0$
- Alternative Hypothesis:
 - The simple linear regression model does fit the data better than the baseline model.
 - $\beta_1 \neq 0$

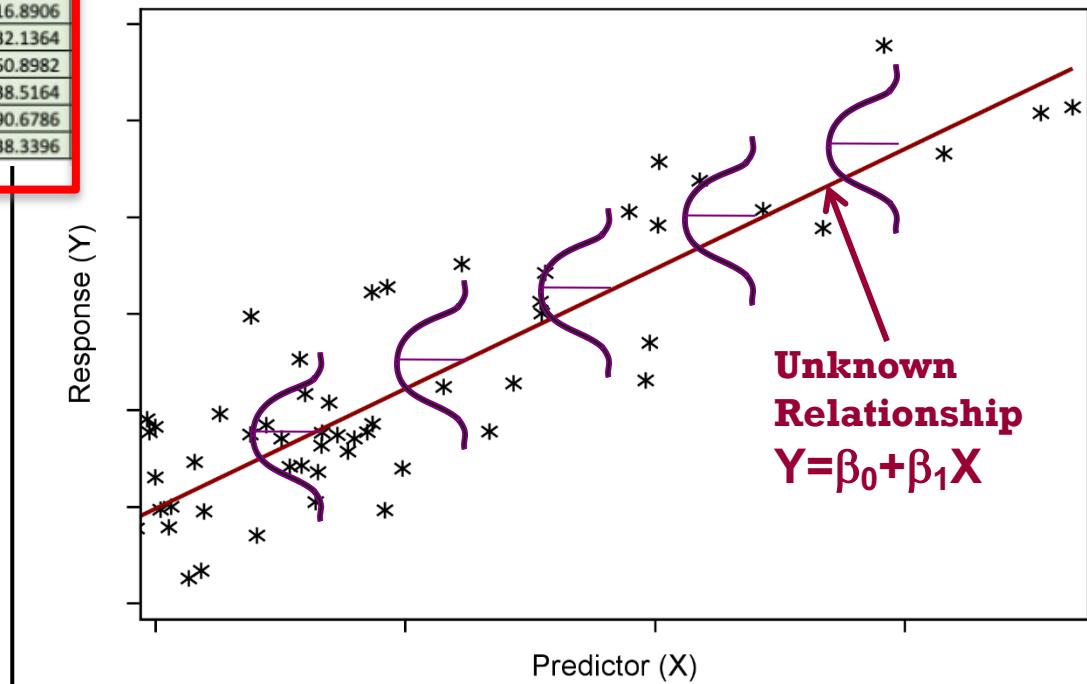


$$F(\text{Model df, Error df}) = MS_M / MS_E$$

Assumptions of Simple Linear Regression

id	chol (x)	bp (y)	predict	error	squared error (SE)
1	437	194	196.1897	(2.1897	4.7948
2	264	121	141.4179	(20.4179	416.8906
3	249	131	136.6689	(5.6689	32.1364
4	297	159	151.8657	7.1343	50.8982
5	243	123	134.7693	(11.7693	138.5164
6	272	161	143.9507	17.0493	290.6786
7	161	115	108.8081	6.1919	38.3396

- The mean of the Ys is accurately modeled by a linear function of the X.
- The random error term, ε , is assumed to have a normal distribution with a mean of zero.
- The random error term, ε , is assumed to have a constant variance, σ^2 .
- Not skew
- The errors are independent.





Multiple Linear Regression

Multiple Linear Regression with Two Variables

- Consider the two-variable model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

- where

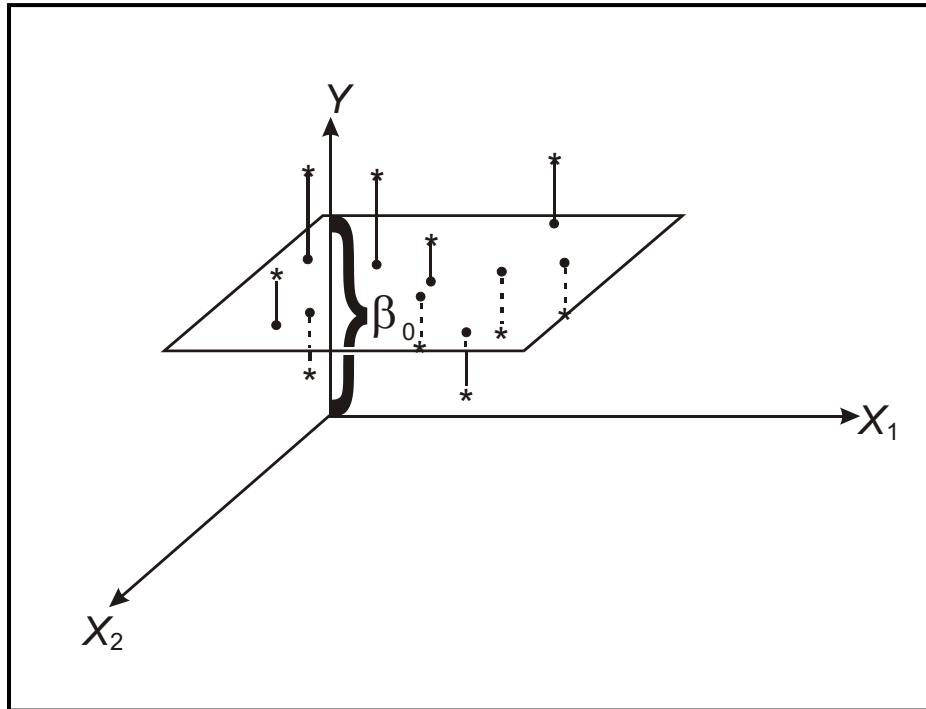
Y is the dependent variable.

X_1 and X_2 are the independent or predictor variables.

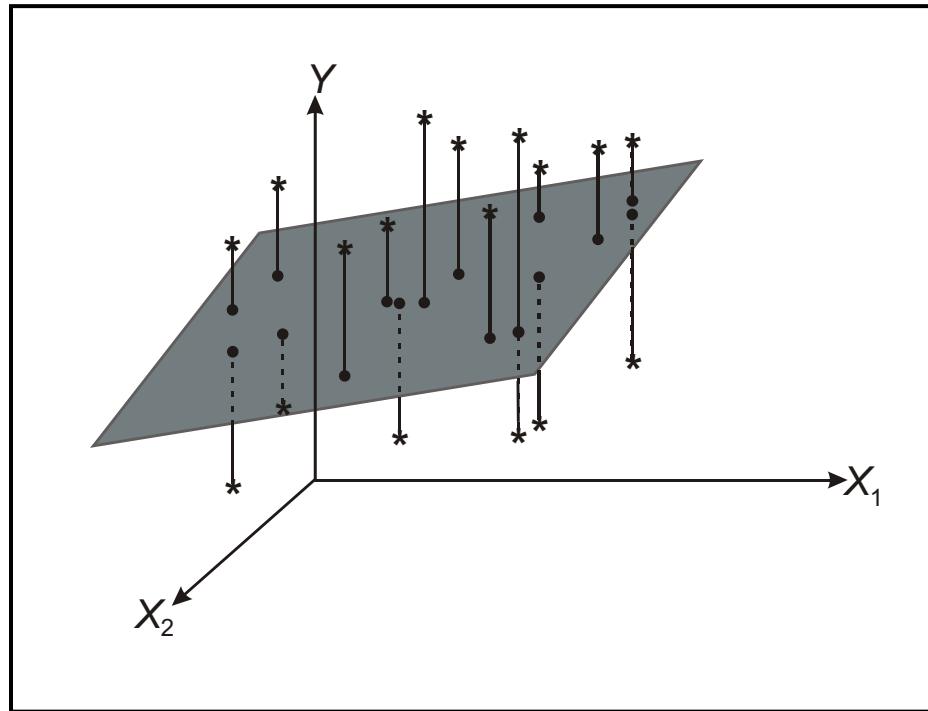
ε is the error term.

β_0 , β_1 , and β_2 are unknown parameters.

Picturing the Model: No Relationship



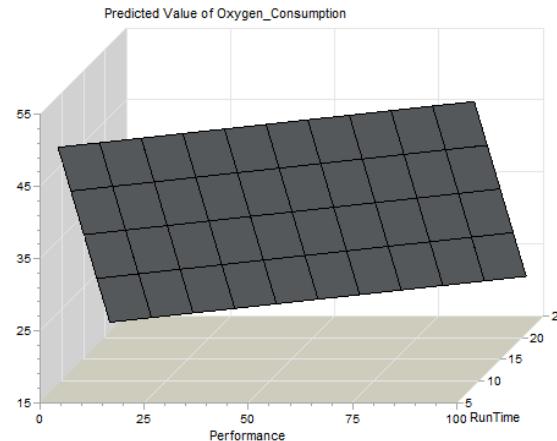
Picturing the Model: A Relationship



The Multiple Linear Regression Model

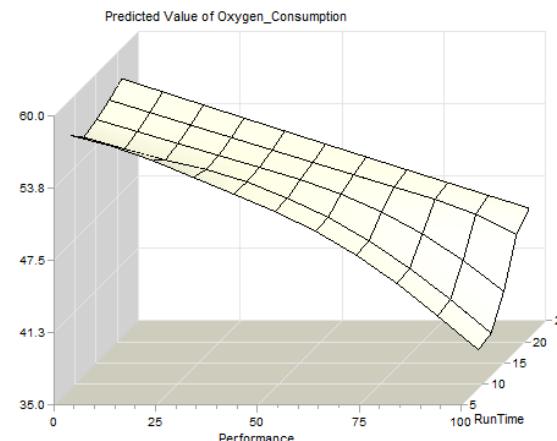
- In general, you model the dependent variable, Y , as a linear function of k independent variables, X_1 through X_k :

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_k X_k + \varepsilon$$



$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

**Linear Model with
only Linear Effects**



$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2 + \varepsilon$$

**Linear Model with
Nonlinear Effects**



The Multiple Linear Regression Model (cont.)

Matrix Multiplication Approach

inputs		target
Age	Income	Spending
25	25,000	400
35	50,000	500
32	35,000	550

$$Y = \beta_0(1) + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

$$\begin{aligned} [Y] &= [X][\beta] \\ [\beta] &= [X]^{-1}[Y] \end{aligned}$$

$$\begin{bmatrix} 400 \\ 500 \\ 550 \end{bmatrix} = \begin{bmatrix} 25 & 25000 \\ 35 & 50000 \\ 32 & 35000 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

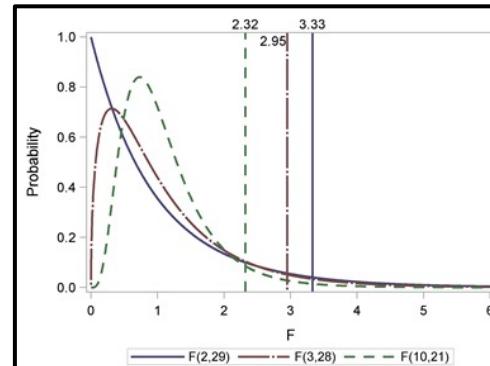
Model Hypothesis Test (F-Test)

■ Null Hypothesis:

- The regression model does not fit the data better than the baseline model.
- $\beta_1 = \beta_2 = \dots = \beta_k = 0$

■ Alternative Hypothesis:

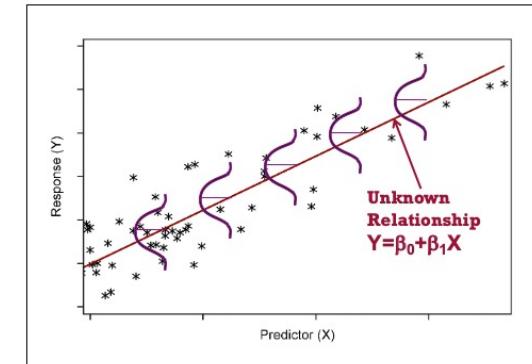
- The regression model does fit the data better than the baseline model.
- Not all β_i s equal zero.



$$F(\text{Model df, Error df}) = MS_M / MS_E$$

Assumptions for Linear Regression

- The mean of the Ys is accurately modeled by a linear function of the X_i .
 - $(y, x_i) = \text{linear relationship (correlation)}$
- The random error term, ε , is assumed to have a normal distribution with a mean of zero.
- The random error term, ε , is assumed to have a constant variance, σ^2 .
 - Not skew
- The errors are independent.



Multiple Linear Regression versus Simple Linear Regression

- **Main Advantage**
- Multiple linear regression enables you to **investigate the relationship** among Y and several independent variables simultaneously.
- **Main Disadvantages**
- Increased complexity makes it **more difficult** to do the following:
 - ascertain which model is “best”
 - interpret the models

Common Applications of Multiple Regression

- Multiple linear regression is a powerful tool for the following tasks:
 - **Prediction** – to develop a model to predict future values of a response variable (Y) based on its relationships with other predictor variables (Xs)
 - **Analytical or Explanatory Analysis** – to develop an understanding of the relationships between the response variable and predictor variables

Prediction

- The terms in the model, the values of their coefficients, and their statistical significance are of secondary importance.
- The focus is on producing a model that is the best at predicting future values of Y as a function of the Xs. The predicted value of Y is given by this formula:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \dots + \hat{\beta}_k X_k$$

Analytical or Explanatory Analysis

- The focus is on understanding the relationship between the dependent variable and the independent variables.
- Consequently, the statistical significance of the coefficients is important as well as the magnitudes and signs of the coefficients.

$$\hat{Y} = \underline{\hat{\beta}_0} + \underline{\hat{\beta}_1}X_1 + \dots + \underline{\hat{\beta}_k}X_k$$

$$R^2 = \frac{SSM}{SST} = 1 - \frac{SSE}{SST} \quad \text{Adjusted R Square}$$

$$R_{ADJ}^2 = 1 - \frac{(n-i)(1-R^2)}{n-p}$$

- $i=1$ if there is an intercept and 0 otherwise
- n =the number of observations used to fit the model
- p =the number of parameters in the model

$$R_{ADJ}^2 = 1 - \frac{(100-1)(1-0.7)}{(100-1)} = 0.70; R^2 = 0.7, n = 100, p = 1$$

$$R_{ADJ}^2 = 1 - \frac{(100-1)(1-0.7)}{(100-3)} = 0.69; R^2 = 0.7, n = 100, p = 3$$

$$R_{ADJ}^2 = 1 - \frac{(100-1)(1-0.7)}{(100-10)} = 0.67; R^2 = 0.7, n = 100, p = 10$$



sklearn.linear_model.LinearRegression

```
class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)
```

[\[source\]](#)

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : bool, optional, default True

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : bool, optional, default False

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before reession by subtracting the mean and dividina by the l2-norm. If you wish to standardize. please use

Methods

`fit(self, X, y[, sample_weight])` Fit linear model.

`get_params(self[, deep])` Get parameters for this estimator.

`predict(self, X)` Predict using the linear model.

`score(self, X, y[, sample_weight])` Return the coefficient of determination R^2 of the prediction.

`set_params(self, **params)` Set the parameters of this estimator.

se.

fficient
cessors.

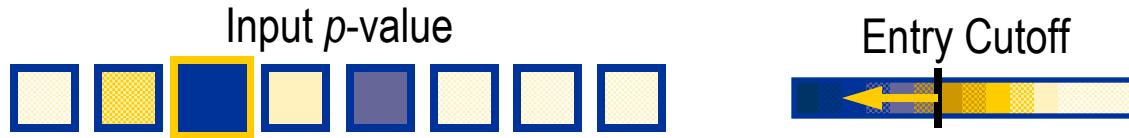


Feature Selection

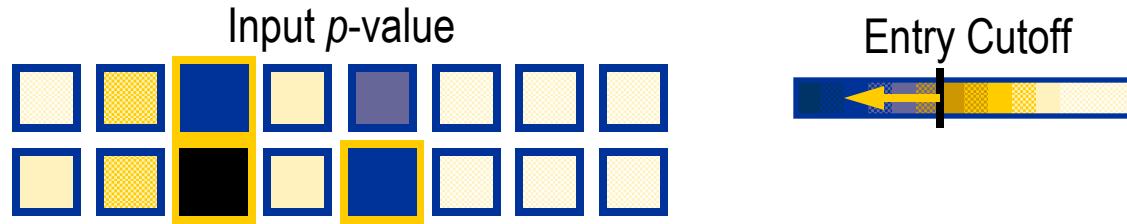
Feature Selection

- Forward
- Backward
- Stepwise

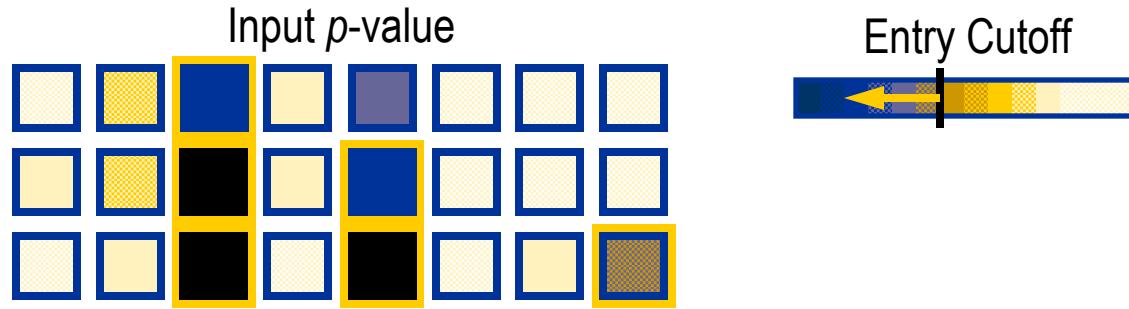
Sequential Selection: Forward



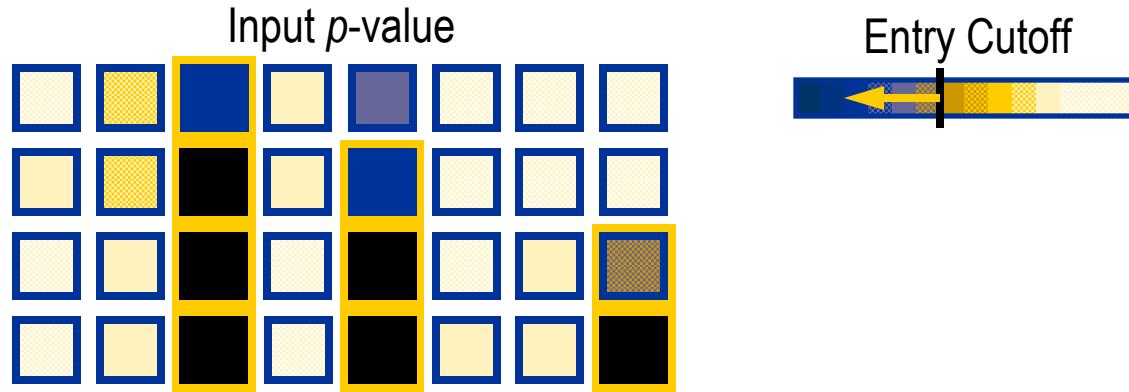
Sequential Selection: Forward



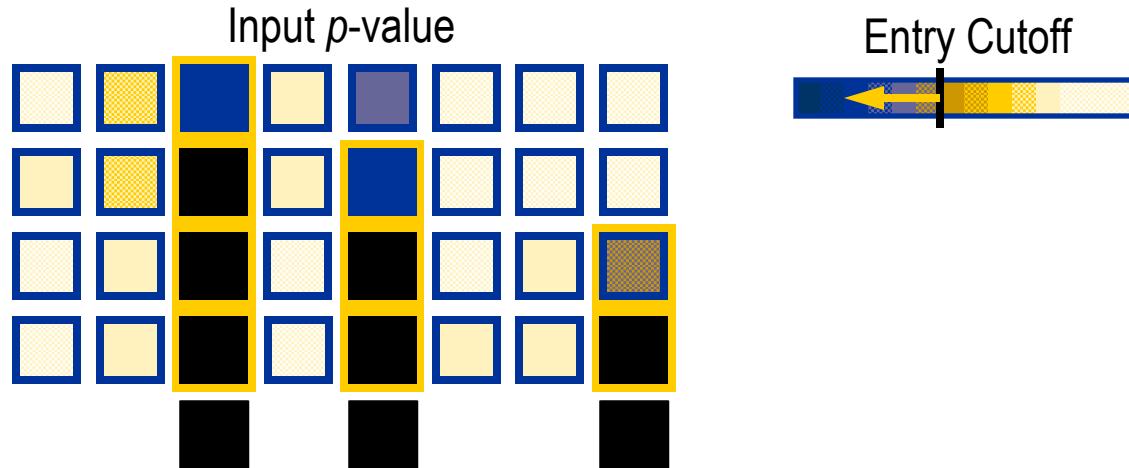
Sequential Selection: Forward



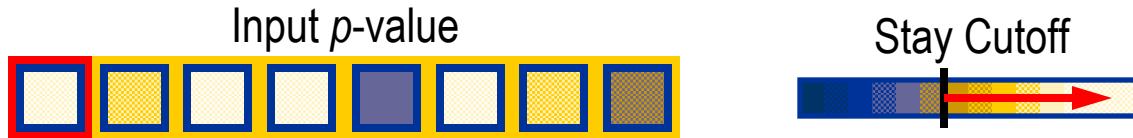
Sequential Selection: Forward



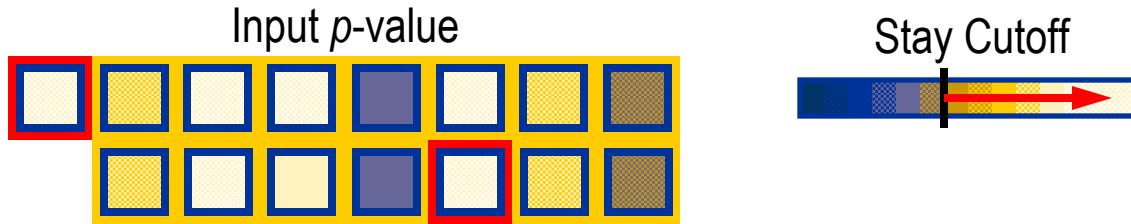
Sequential Selection: Forward



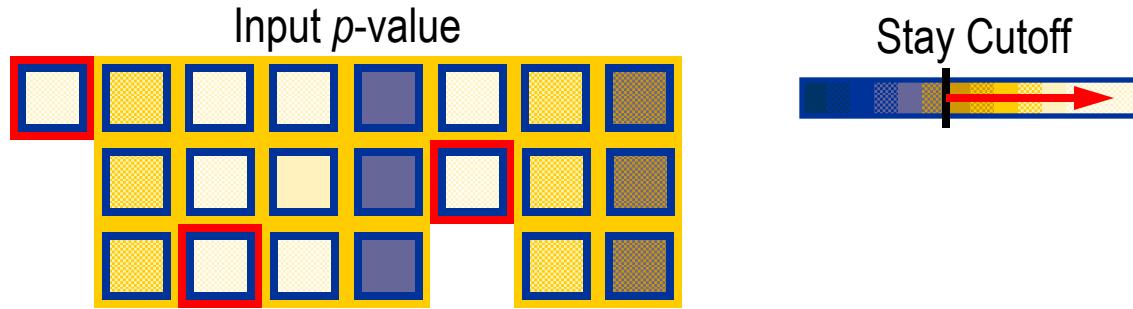
Sequential Selection: Backward



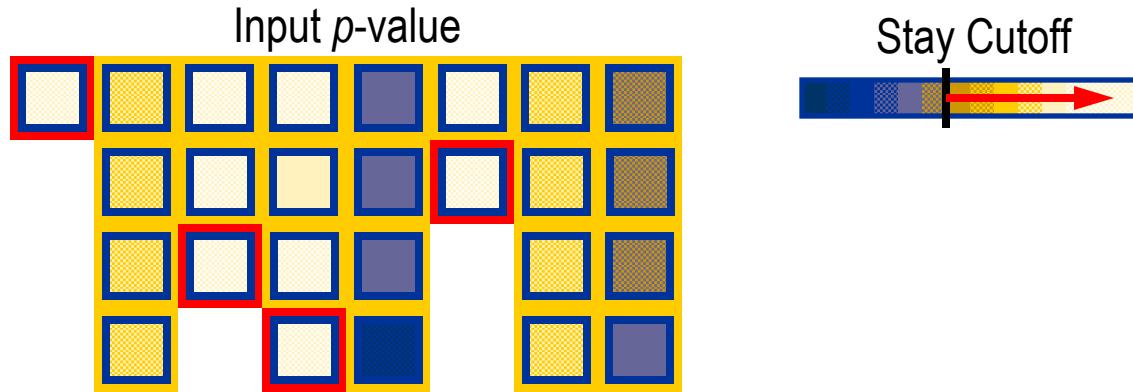
Sequential Selection: Backward



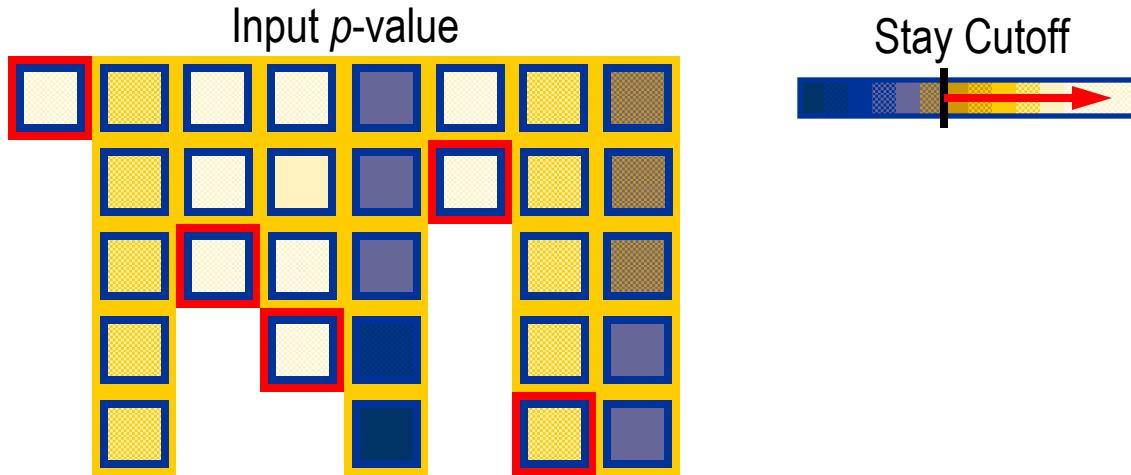
Sequential Selection: Backward



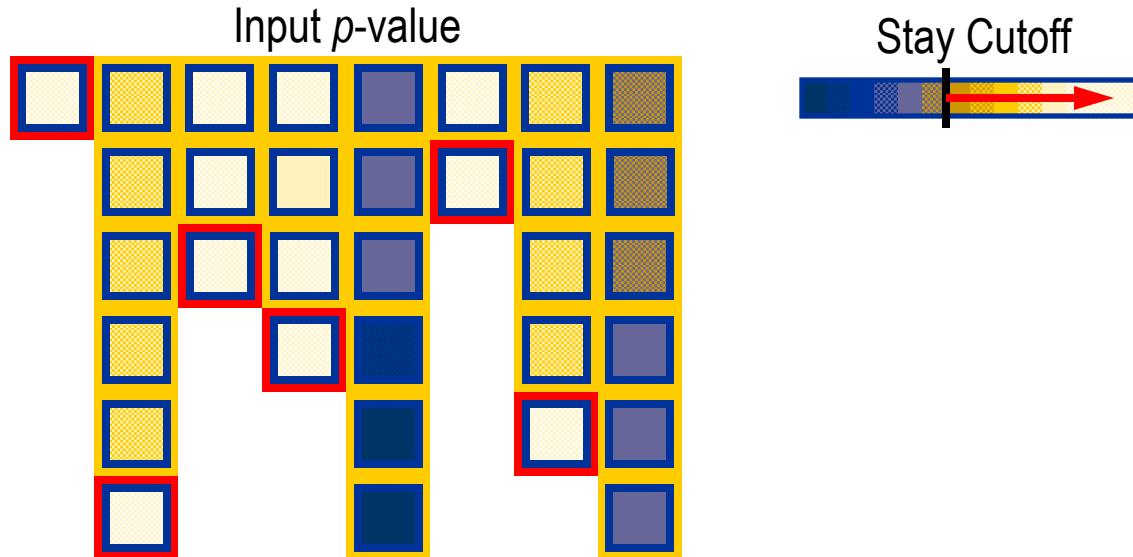
Sequential Selection: Backward



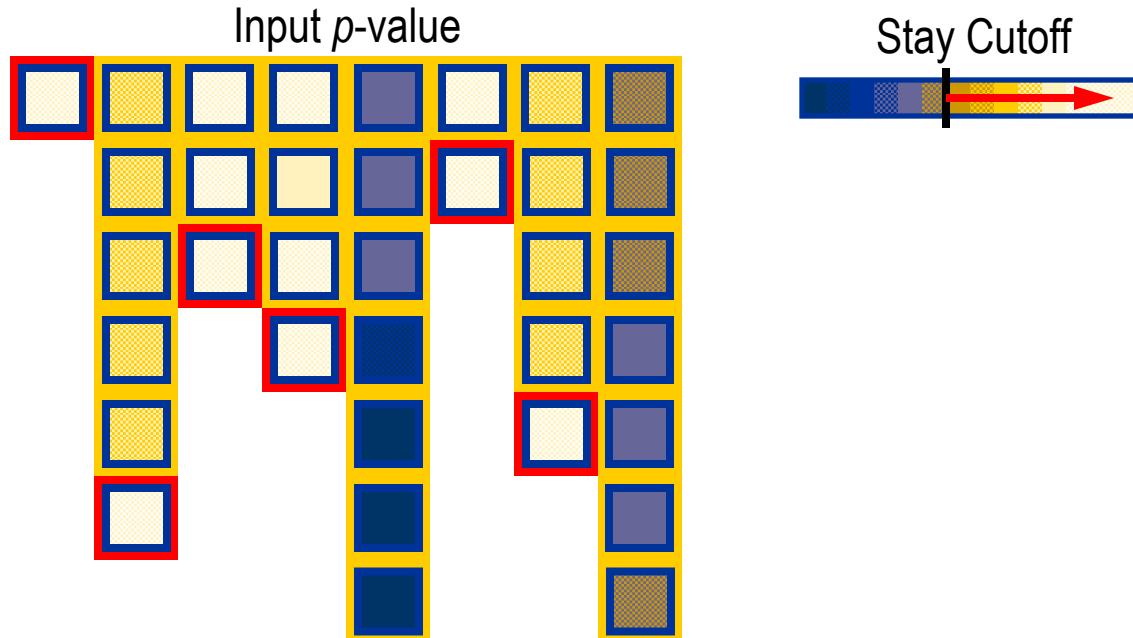
Sequential Selection: Backward



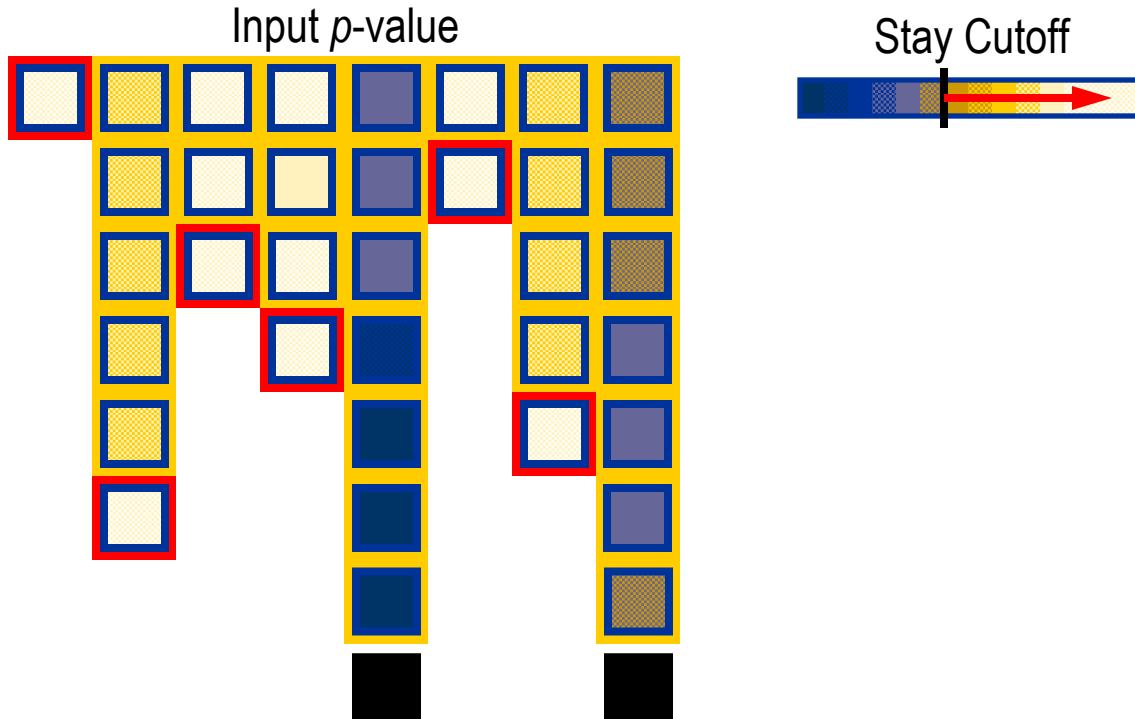
Sequential Selection: Backward



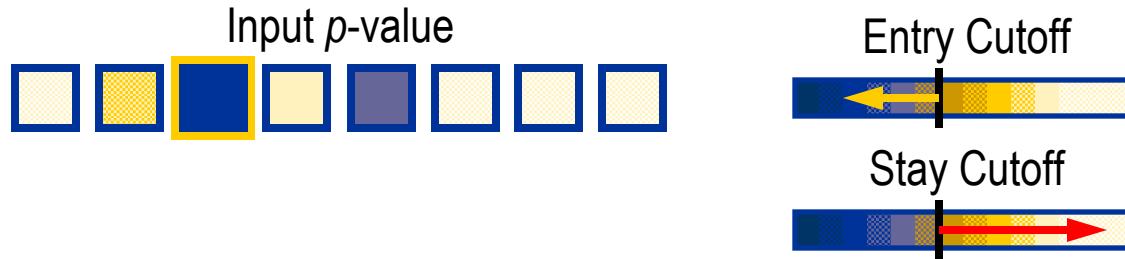
Sequential Selection: Backward



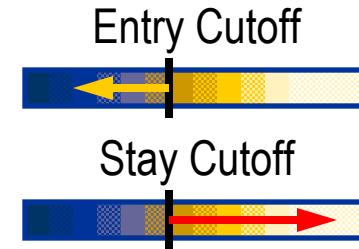
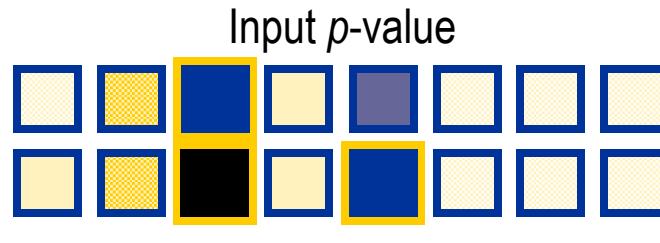
Sequential Selection: Backward



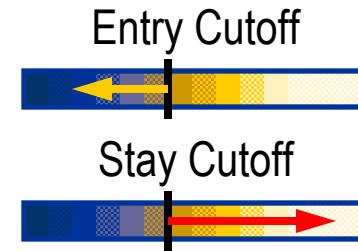
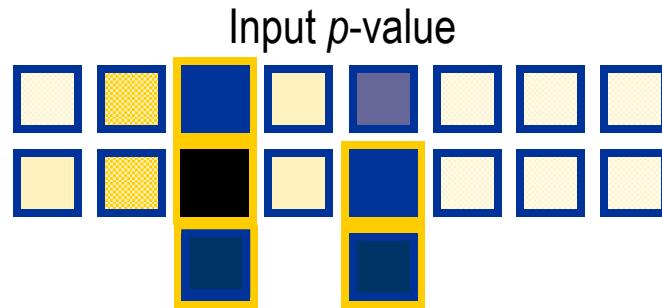
Sequential Selection: Stepwise



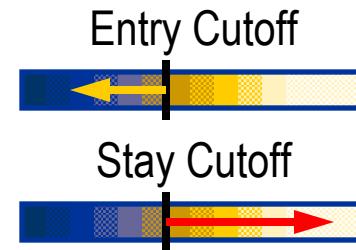
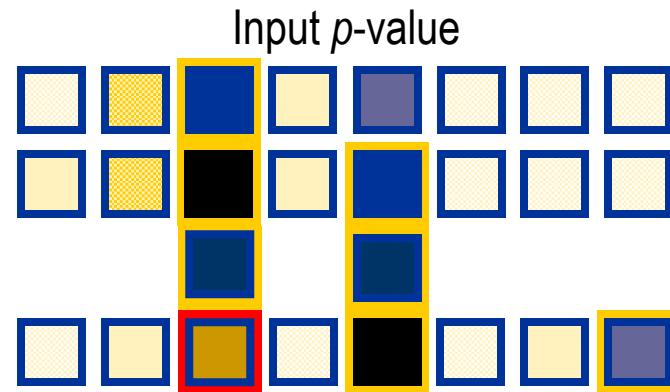
Sequential Selection: Stepwise



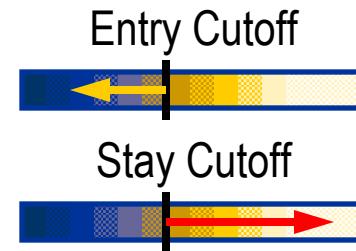
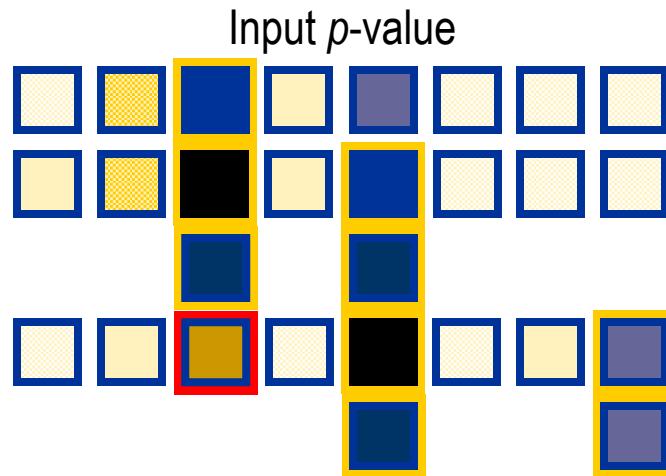
Sequential Selection: Stepwise



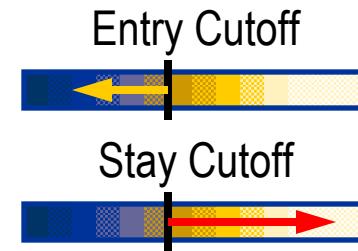
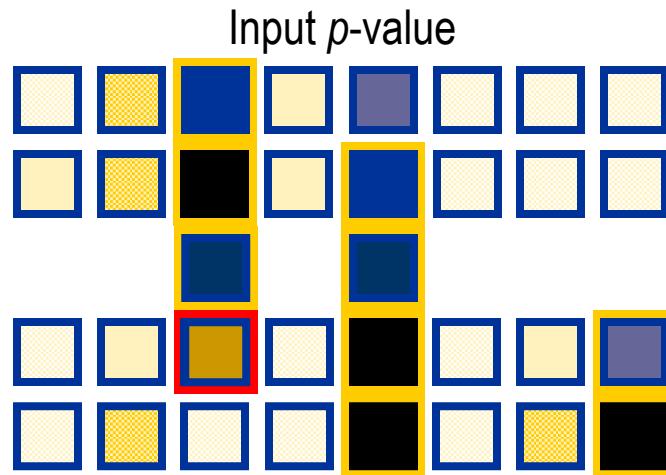
Sequential Selection: Stepwise



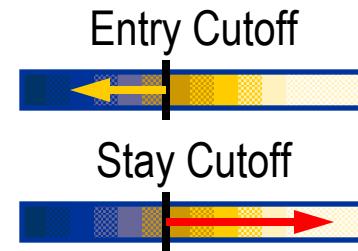
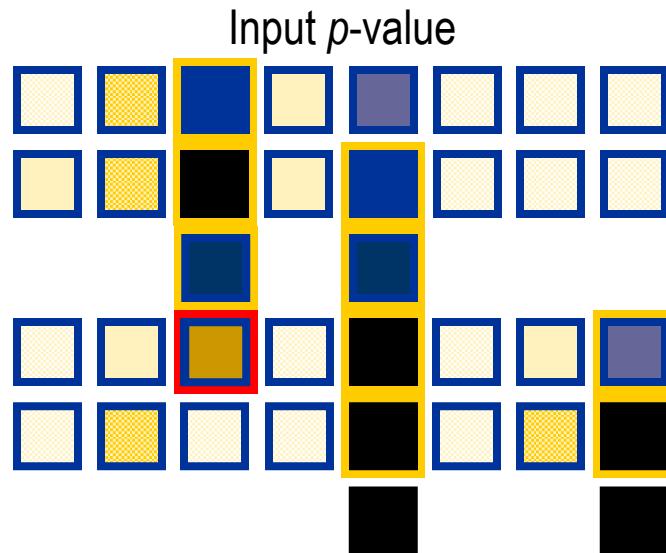
Sequential Selection: Stepwise



Sequential Selection: Stepwise



Sequential Selection: Stepwise





Sequential Feature Selector

Overview

Example 1 - A simple Sequential Forward Selection example

Example 2 - Toggling between SFS, SBS, SFFS, and SBFS

Sequential Feature Selector

Implementation of *sequential feature algorithms* (SFAs) -- greedy search algorithms -- that have been developed as a suboptimal solution to the computationally often not feasible exhaustive search.

```
from mlxtend.feature_selection import SequentialFeatureSelector
```

- Sequential feature selection algorithms are a family of **greedy** search algorithms that are used to reduce an initial d -dimensional feature space to a k -dimensional feature subspace where $k < d$.
- In a nutshell, SFAs remove or add **one feature at the time** based on **the classifier performance (such as, accuracy)** until a feature subset of the desired size k is reached.
- Sequential Forward Selection (SFS)
- Sequential Backward Selection (SBS)
- Sequential Forward Floating Selection (SFFS)
- Sequential Backward Floating Selection (SBFS)

- **scoring** : str, callable, or None (default: None)

If None (default), uses 'accuracy' for sklearn classifiers and 'r2' for sklearn regressors. If str, uses a sklearn scoring metric string identifier, for example {accuracy, f1, precision, recall, roc_auc} for classifiers, {'mean_absolute_error', 'mean_squared_error', 'neg_mean_squared_error', 'median_absolute_error', 'r2'} for regressors. If a callable object or function is provided, it has to be conform with sklearn's signature `scorer(estimator, X, y)`; see http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html for more information.

- **cv** : int (default: 5)

Integer or iterable yielding train, test splits. If cv is an integer and `estimator` is a classifier (or y consists of integer class labels) stratified k-fold. Otherwise regular k-fold cross-validation is performed. No cross-validation if cv is None, False, or 0.



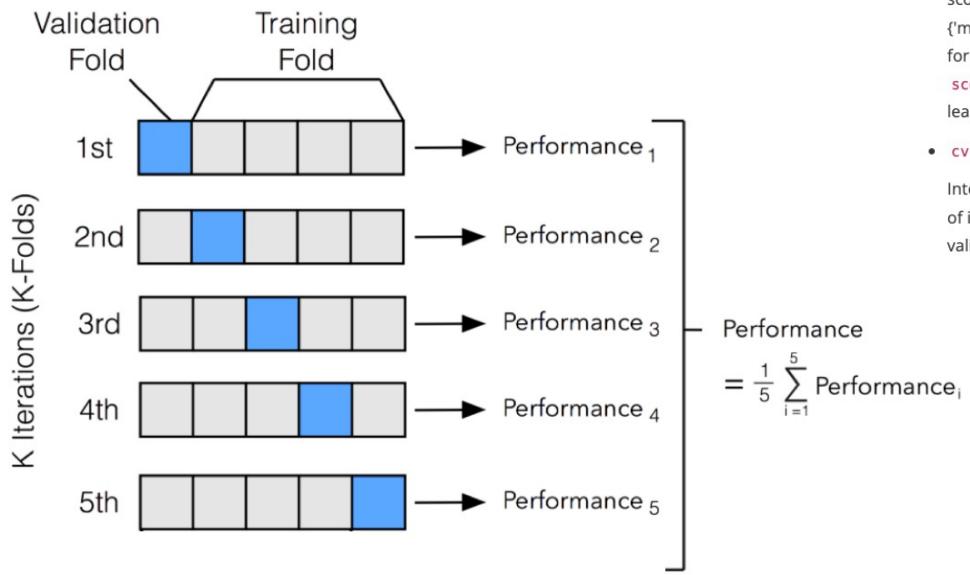
Seqeuntial Feature Selector (cont.)

- The ***floating*** variants, SFFS and SBFS, can be considered as extensions to the simpler SFS and SBS algorithms.
- The dimensionality of the subset during the search can be thought to be “**floating**” **up and down**. (somewhat similar to **stepwise**)
 - Sequential Forward Floating Selection (SFFS)
 - Sequential Backward Floating Selection (SBFS)

```

1 # Build RF classifier to use in feature selection
2 clf = LogisticRegression()
3
4 # Build step forward feature selection
5 sfs1 = sfs(clf,
6             k_features=5,
7             forward=True,
8             floating=False,
9             verbose=2,
10            scoring='accuracy',
11            cv=5)
12
13 # Perform SFFS
14 sfs1 = sfs1.fit(bank_final, y)

```



```

1 # Which features?
2 feat_cols = list(sfs1.k_feature_idx_)
3 print(feat_cols)

[6, 10, 15, 16, 17]

```

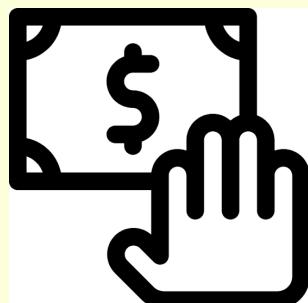
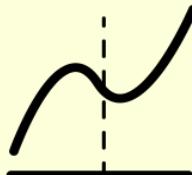
- **scoring** : str, callable, or None (default: None)

If None (default), uses 'accuracy' for sklearn classifiers and 'r2' for sklearn regressors. If str, uses a sklearn scoring metric string identifier, for example (accuracy, f1, precision, recall, roc_auc) for classifiers, {'mean_absolute_error', 'mean_squared_error', 'neg_mean_squared_error', 'median_absolute_error', 'r2'} for regressors. If a callable object or function is provided, it has to be conform with sklearn's signature `scorer(estimator, X, y)`; see http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html for more information.

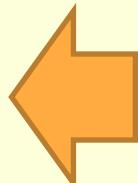
- **cv** : int (default: 5)

Integer or iterable yielding train, test splits. If cv is an integer and `estimator` is a classifier (or y consists of integer class labels) stratified k-fold. Otherwise regular k-fold cross-validation is performed. No cross-validation if cv is None, False, or 0.

Logistic Regression



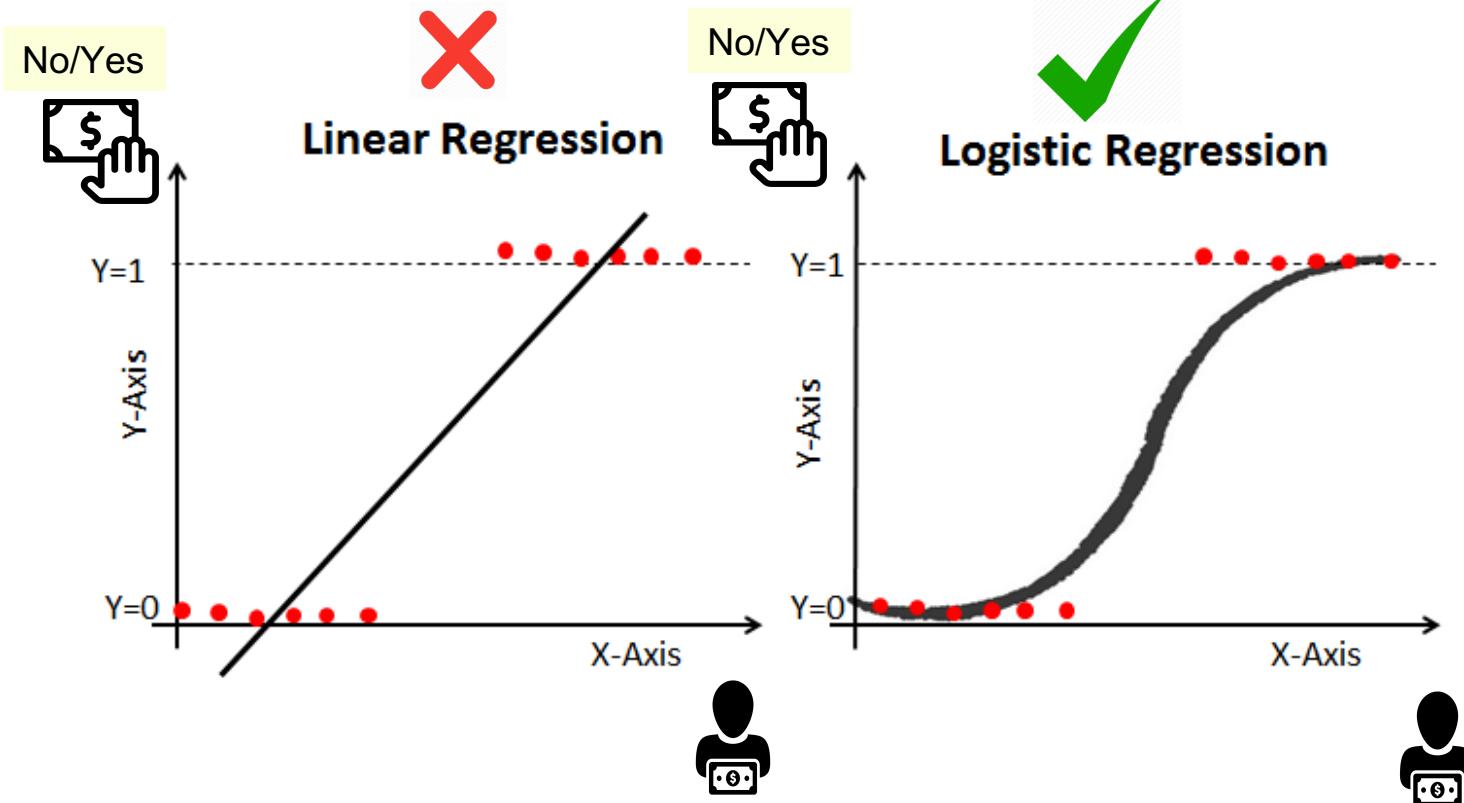
Purchase
(No/Yes)



Salary

Type of Predictors	Categorical	Continuous	Continuous and Categorical
Type of Response			
Continuous	Analysis of Variance (ANOVA)	Ordinary Least Squares (OLS) Regression	Analysis of Covariance (ANCOVA)
Categorical	Contingency Table Analysis or Logistic Regression	Logistic Regression	Logistic Regression

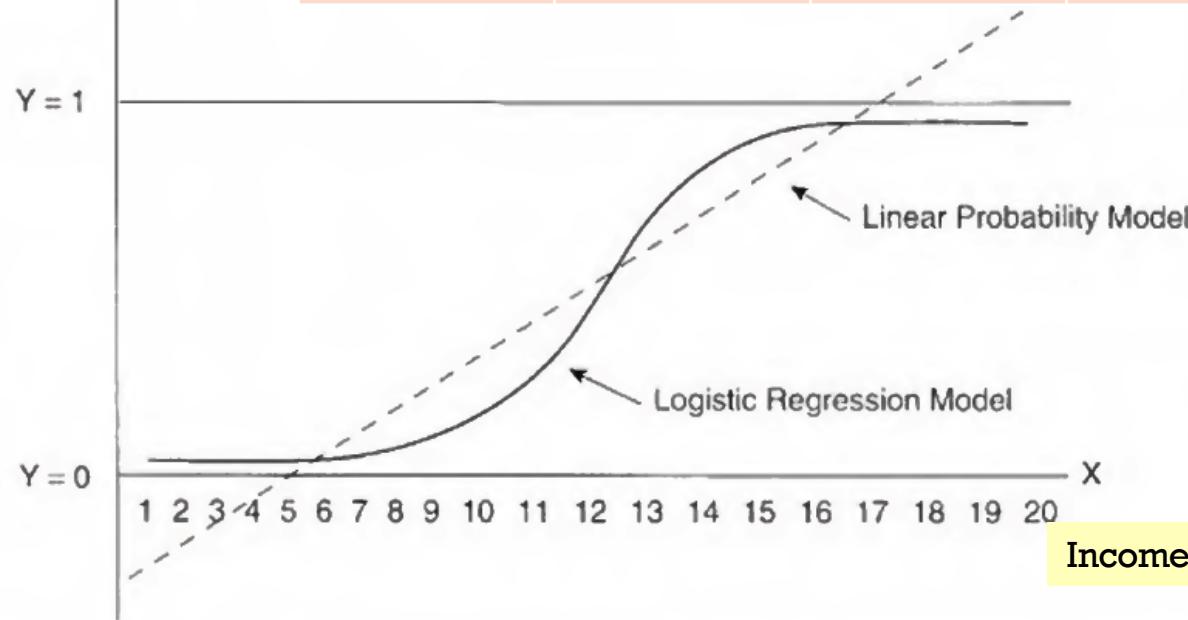
Logistic Regression





Classification Task

Prob. of purchase	Age	Income	Gender	Province	Purchase
Y	25	25,000	Female	Bangkok	Yes (1)
	35	50,000	Female	Nontaburi	Yes (1)
	32	35,000	Male	Bangkok	No (0)

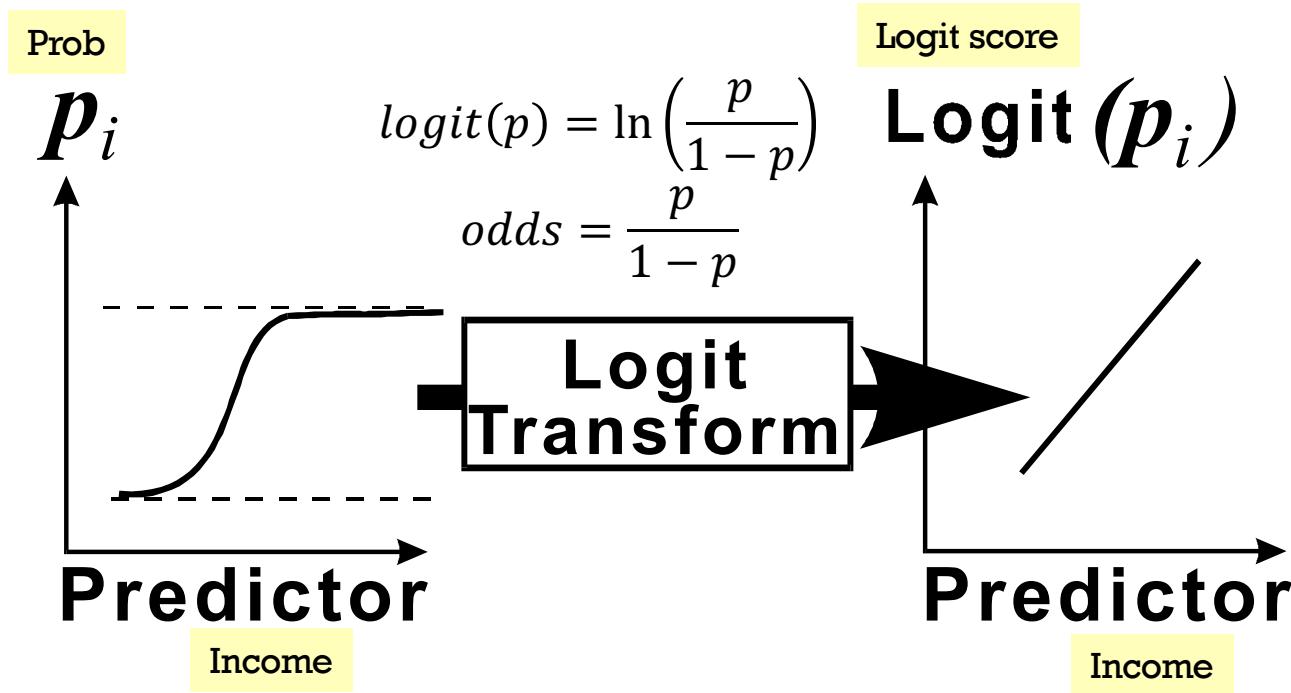


Logistic function
Sigmoid function



Logistic Regression

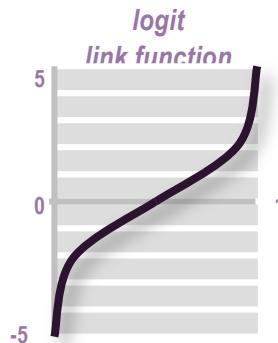
Logit link function: Non-linear to Linear



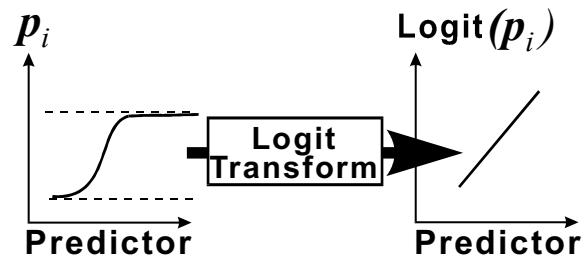


Logistic Regression Prediction Formula

$$\log \left(\frac{\hat{p}}{1 - \hat{p}} \right) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2 \quad \text{logit scores}$$



The logit link function transforms probabilities (between 0 and 1) to logit scores (between $-\infty$ and $+\infty$).



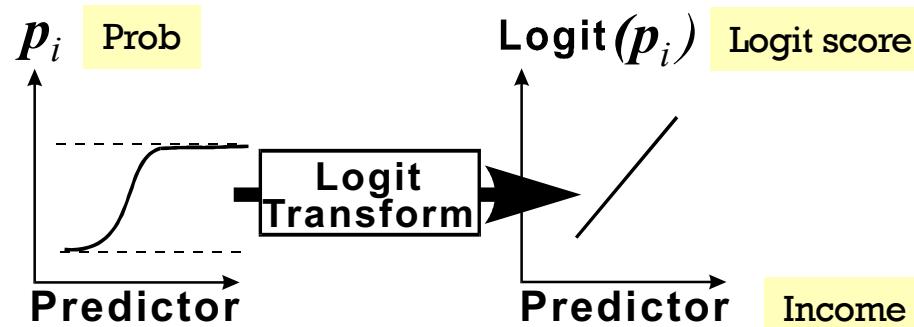


Logit Link Function

$$\log\left(\frac{\hat{p}}{1 - \hat{p}}\right) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2 = \text{logit}(\hat{p})$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

To obtain prediction estimates, the logit equation is solved for \hat{p} .





Training Phase: Regressions

Maximum Log-Likelihood Estimation (MLE)

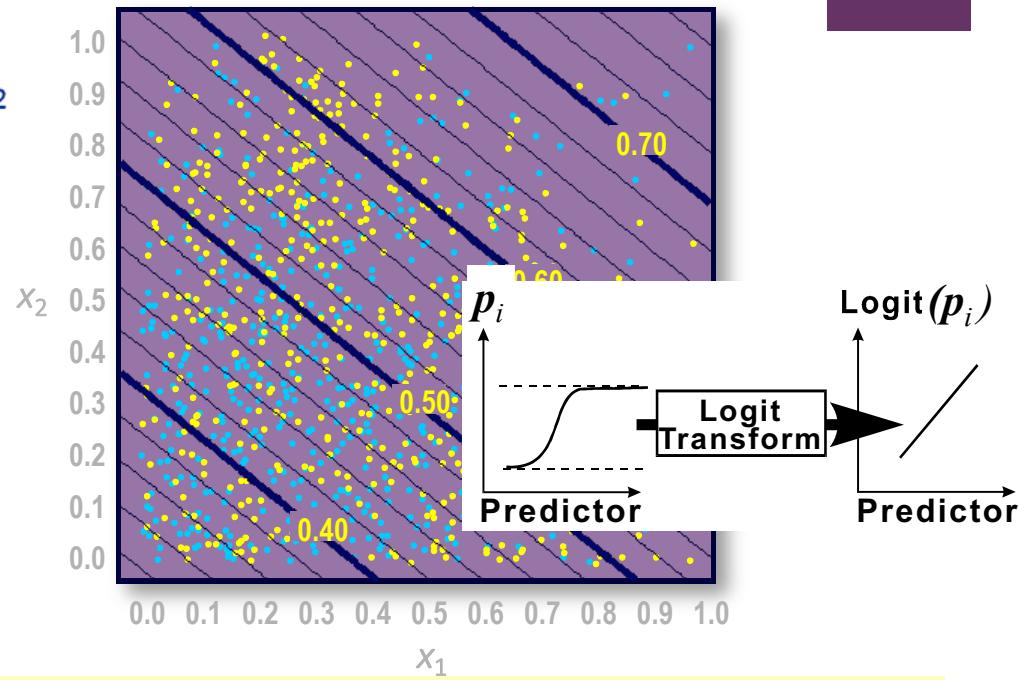
$$\text{logit}(\hat{p}) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Find parameter estimates
by *maximizing*

$$\sum_{\text{primary outcome training cases}} \log(\hat{p}_i) + \sum_{\text{secondary outcome training cases}} \log(1 - \hat{p}_i)$$

log-likelihood function



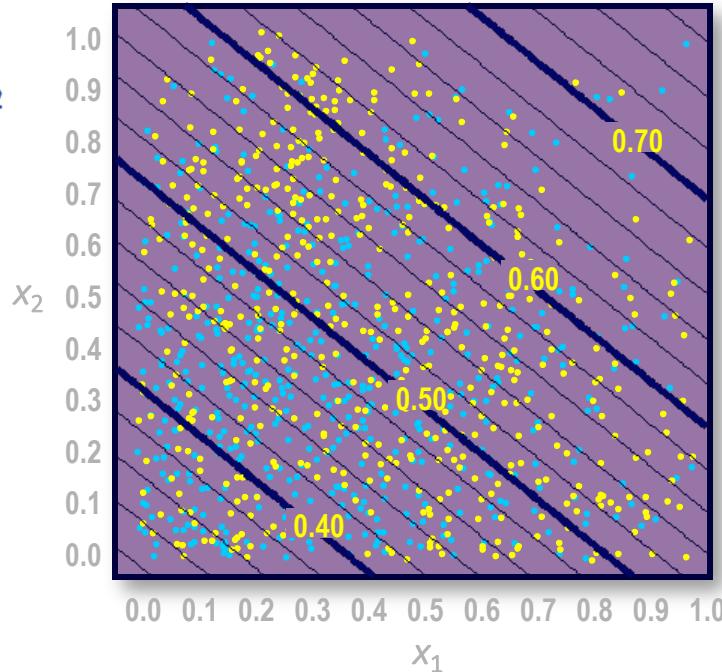
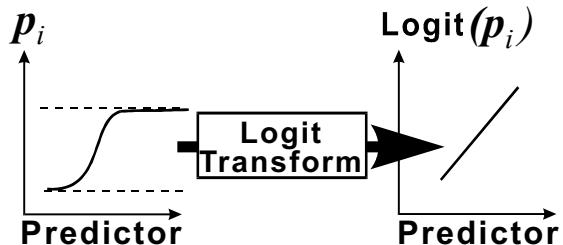
For each set of parameters (w_0, w_1, w_2), we can calculate log-likelihood (LL) of the whole training data.
So, pick the set of parameters with a maximum LL (MLE).

Testing Phase: Regressions

$$\text{logit}(\hat{p}) = -0.81 + 0.92x_1 + 1.11x_2$$

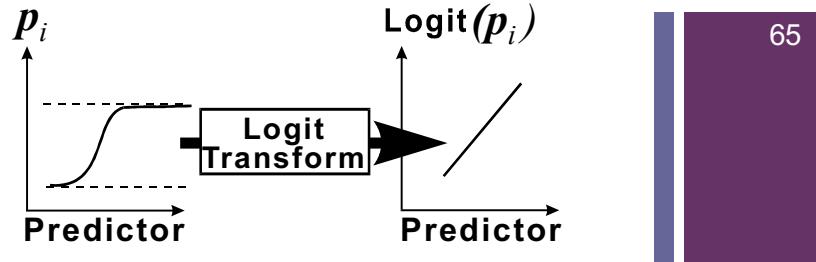
$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Using the maximum likelihood estimates, the prediction formula assigns a logit score to each x_1 and x_2 .





Quiz

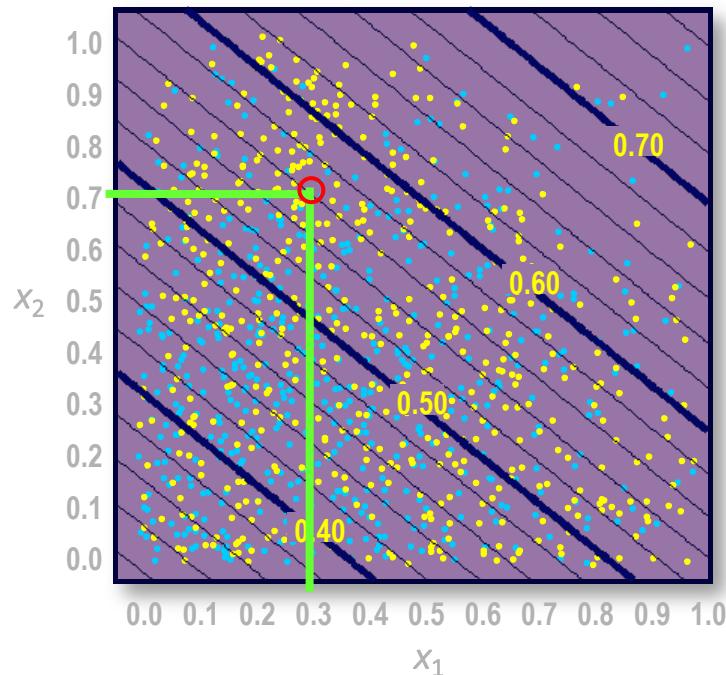


■ What is the logistic regression prediction for the indicated point?

- a. 0.243
- b. 0.56
- c. yellow
- d. It depends.

$$\text{logit}(\hat{p}) = -0.81 + 0.92x_1 + 1.11x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$





Training Phase: Regressions (Recap)

Maximum Log-Likelihood Estimation (MLE)

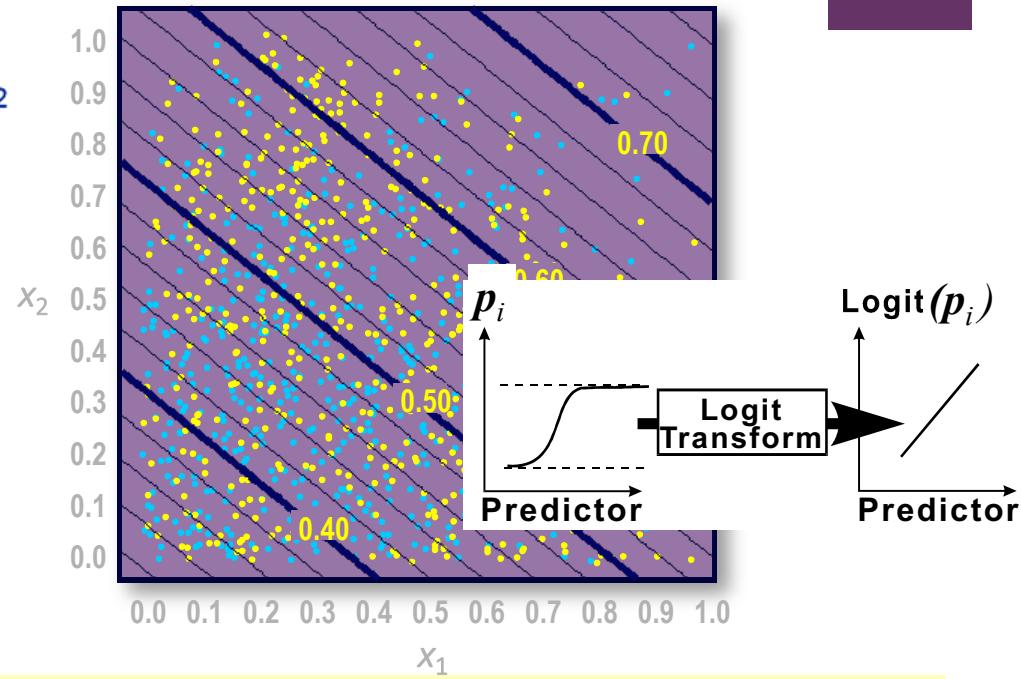
$$\text{logit}(\hat{p}) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Find parameter estimates by *maximizing*

$$\sum_{\text{primary outcome training cases}} \log(\hat{p}_i) + \sum_{\text{secondary outcome training cases}} \log(1 - \hat{p}_i)$$

log-likelihood function



For each set of parameters (w_0, w_1, w_2), we can calculate log-likelihood (LL) of the whole training data. So, pick the set of parameters with a maximum LL (MLE).

$$P(Y = y) = p^y(1 - p)^{1-y}, y = 0, 1$$

$$P(Y = 1) = p^1(1 - p)^0 = p$$

$$P(Y = 0) = p^0(1 - p)^{1-0} = (1 - p)$$

$$Likelihood = \prod_{i=1}^n p^{y_i}(1 - p)^{1-y_i}$$

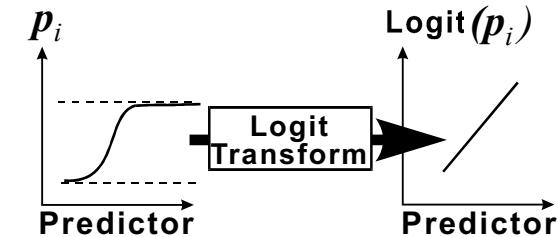
$$Log-Likelihood (LL) = \sum_{i=1}^n y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

Outcome	Predicted Probability		
	y_i	p_i	$(1 - p_i)$
	1	.899	.101
	1	.540	.460
	0	.317	.683
	1	.516	.439
	1	.698	.302
	0	.457	.543
	0	.234	.766
	1	.899	.101
	1	.451	.439
	1	.764	.236
	0	.457	.543
	0	.561	.439
	1	.698	.302
	1	.457	.543
	0	.899	.101

$$\text{logit}(\hat{p}) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

$$\text{logit}(\hat{p}) = -0.81 + 0.92 x_1 + 1.11 x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$



$$-2 \cdot \text{log-likelihood} = 17.48.$$

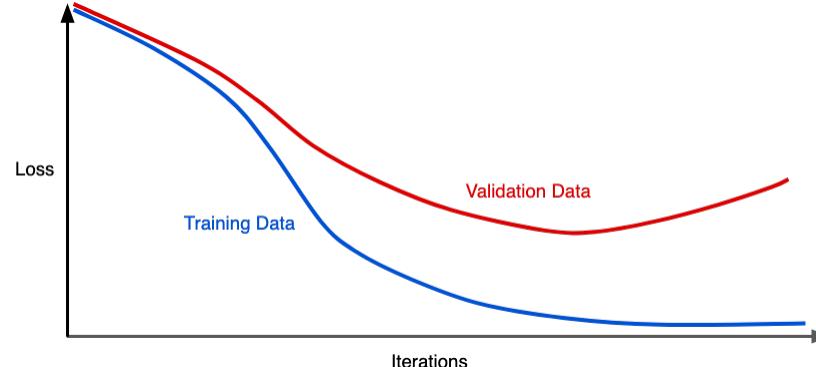
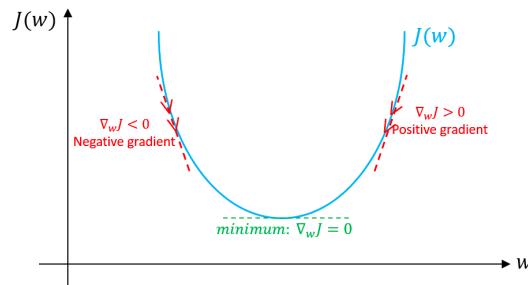
-2LL can be considered as error (loss), so smaller is better.

MLE Solver

$$\text{logit}(\hat{p}) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

$$\text{logit}(\hat{p}) = -0.81 + 0.92 x_1 + 1.11 x_2$$

- Maximum Log-Likelihood Estimation (MLE) refers to **maximize LL**
- Or **minimize -2LL (loss)**
- Gradient descent optimization
 - Stochastic Average Gradient solver



[Prev](#) [Up](#) [Next](#)**scikit-learn 0.22.1**[Other versions](#)

Please [cite us](#) if you use the software.

sklearn.linear_model.LogisticRegression

Examples using

[sklearn.linear_model.LogisticRe](#)[Prev](#) [Up](#) [Next](#)**scikit-learn 0.10.2**[Other versions](#)

Please [cite us](#) if you use the software.

sklearn.linear_model.LogisticRegression

Examples using

[sklearn.linear_model.LogisticRe](#)

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

[\[source\]](#)

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

solver : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=‘lbfgs’

Algorithm to use in the optimization problem.

- For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.
- For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.
- ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’ handle L2 or no penalty
- ‘liblinear’ and ‘saga’ also handle L1 penalty
- ‘saga’ also supports ‘elasticnet’ penalty
- ‘liblinear’ does not support setting `penalty='none'`

Note that ‘sag’ and ‘saga’ fast convergence is only guaranteed on features with approximately the same scale.

You can preprocess the data with a scaler from `sklearn.preprocessing`.

New in version 0.17: Stochastic Average Gradient descent solver.

New in version 0.19: SAGA solver.

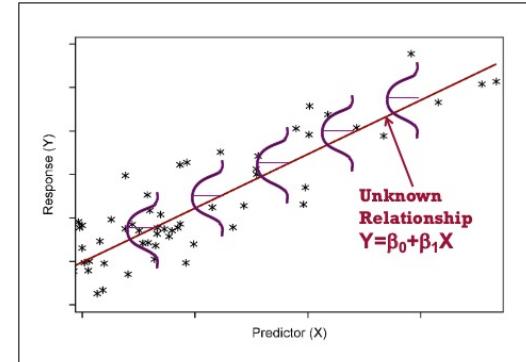
Changed in version 0.22: The default solver changed from ‘liblinear’ to ‘lbfgs’ in 0.22.

max_iter : int, default=100

Maximum number of iterations taken for the solvers to converge.

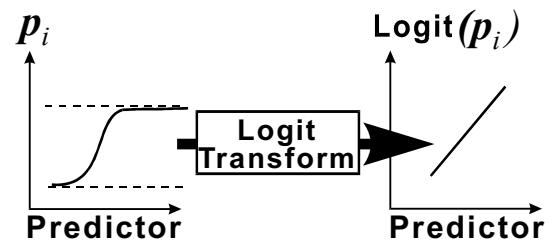
Assumptions for Linear Regression (recap)

- The mean of the Ys is accurately modeled by a linear function of the X_i .
 - $(y, x_i) = \text{linear relationship (correlation)}$
- The random error term, ε , is assumed to have a normal distribution with a mean of zero.
- The random error term, ε , is assumed to have a constant variance, σ^2 .
 - Not skew
- The errors are independent.



Assumptions for Logistic Regression

- The mean of the logit is accurately modeled by a linear function of the X_i .
 - $(\text{logit}, x_i) = \text{linear relationship}$
- The random error term, ϵ , is assumed to have a normal distribution with a mean of zero.
- The random error term, ϵ , is assumed to have a constant variance, σ^2 .
 - Not skew
- The errors are independent.



Beyond the prediction

- Manage missing values
- Interpret the model
- Handle outliers
- Use nonnumeric inputs

Missing Values and Regression Modeling

Training Data

Problem 1: Training data cases with missing values on inputs used by a regression model are ignored.



Missing Values and Regression Modeling

Training Data

inputs							target

Consequence: Missing values can significantly reduce your amount of training data for regression modeling!



Missing Values and the Prediction Formula

$$\text{logit}(\hat{p}) = -0.81 + 0.92 \cdot x_1 + 1.11 \cdot x_2$$

Predict: $(x_1, x_2) = (0.3, ?)$

Problem 2: Prediction formulas cannot score cases with missing values.



Missing Values and the Prediction Formula

$$\text{logit}(\hat{p}) = -0.81 + 0.92 \cdot 0.3 + 1.11 \cdot ?$$

Predict: $(x_1, x_2) = (0.3, ?)$

Problem 2: Prediction formulas cannot score cases with missing values.



Interpret the model: Positive effect (income)

Recap how to interpret “regression”

$$\text{logit}(p) = 500 + 0.2 \times \text{Income} - 0.8 \times \text{Age}$$

$$\ln(\text{odds}) = 500 + 0.2 \times \text{Income} - 0.8 \times \text{Age}$$

$$\ln(\text{odds}(\text{Income} = 1)) = 0.2 \times 1 = 0.2$$

$$\text{odds}(\text{Income} = 1) = e^{0.2} = 1.22$$

$$\ln(\text{odds}(\text{Income} = 0)) = 0.2 \times 0 = 0$$

$$\text{odds}(\text{Income} = 0) = e^0 = 1$$

$$\text{Odds Ratio}(\text{Income}) = \frac{\text{odds}(\text{Income} = 1)}{\text{odds}(\text{Income} = 0)} = \frac{1.22}{1} = 1.22 (+22\%) = \exp(w_1)$$

- OR(Income) = 1.22 (22% increase)
- When income increases by 1THB, it has **higher** chance (odds) to purchase for 22%



Interpret the model: Negative effect (age)

Recap how to interpret “regression”

$$\text{logit}(p) = 500 + 0.2 \times \text{Income} - 0.8 \times \text{Age}$$

$$\ln(\text{odds}) = 500 + 0.2 \times \text{Income} - 0.8 \times \text{Age}$$

$$\ln(\text{odds}(\text{Age} = 1)) = -0.8 \times 1 = -0.8$$

$$\text{odds}(\text{Age} = 1) = e^{-0.8} = 0.45$$

$$\ln(\text{odds}(\text{Age} = 0)) = -0.8 \times 0 = 0$$

$$\text{odds}(\text{Age} = 0) = e^0 = 1$$

$$\text{Odds Ratio}(\text{Age}) = \frac{\text{odds}(\text{Age} = 1)}{\text{odds}(\text{Age} = 0)} = \frac{0.45}{1} = 0.45 \text{ (-55\%)} = \exp(w_1)$$

- OR(Age) = 0.45 (55% decrease)
- When age increases by 1 year, it has **lower** chance (odds) to purchase for 55%.

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None) ¶
```

[\[source\]](#)

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi_class’ option is set to ‘ovr’, and uses the cross-entropy loss if the ‘multi_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’, ‘saga’ and ‘newton-cg’ solvers.)

This class implements regularized logistic regression using **regularization is applied by default**. It can handle both dense and sparse input formats, and it will use 64-bit floats for optimal performance; any other input format will be converted to 64-bit floats.

The ‘newton-cg’, ‘sag’, and ‘lbfgs’ solvers support only L2 regularization. The ‘saga’ solver supports both L1 and L2 regularization, with a dual coordinate descent solver. The ‘liblinear’ solver supports both L1 and L2 regularization, with a dual coordinate descent solver supported by the ‘saga’ solver.

Read more in the [User Guide](#).

Examples

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Parameters:

penalty : {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}

Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only L2 penalties. ‘elasticnet’ is only supported by the ‘saga’ solver. If ‘none’ (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing ‘multinomial’ + L1)

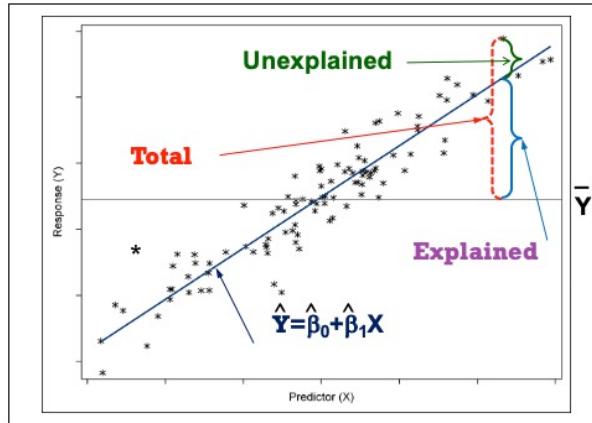


Evaluation

Pseudo R²

TP, TN, FP, FN

Precision, Recall, F1



R² (recap)

$$R^2 = \frac{SSM}{SST} = 1 - \frac{SSE}{SST}$$

id	chol (x)	bp (y)	predict	error	squared error (SE)	guess	(y - y_bar)	squared total (ST)
1	437	194	196.1897	(2.1897)	4.7948	143.4286	50.5714	2,557.4694
2	264	121	141.4179	(20.4179)	416.8906	143.4286	(22.4286)	503.0408
3	249	131	136.6689	(5.6689)	32.1364	143.4286	(12.4286)	154.4694
4	297	159	151.8657	7.1343	50.8982	143.4286	15.5714	242.4694
5	243	123	134.7693	(11.7693)	138.5164	143.4286	(20.4286)	417.3265
6	272	161	143.9507	17.0493	290.6786	143.4286	17.5714	308.7551
7	161	115	108.8081	6.1919	38.3396	143.4286	(28.4286)	808.1837
average	274.7143	143.4286	SSE		972.2548	SST		4,991.7143
			MSF		138.8935			
			RMSE		11.7853			
	R^2	1 - (SSE/SST)	0.8052					



Pseudo R²

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2}$$

Outcome	Predicted Probability	
	\hat{Y}	$1 - \hat{Y}$
1	.899	.101
1	.540	.460
0	.317	.683
1	.516	.439
1	.698	.302
0	.457	.543
0	.234	.766
1	.899	.101
1	.451	.439
1	.764	.236
0	.457	.543
0	.561	.439
1	.698	.302
1	.457	.543
0	.899	.101

$$\bar{y} = 0.6$$

$$^{a}-2 * \text{log-likelihood} = 17.48.$$



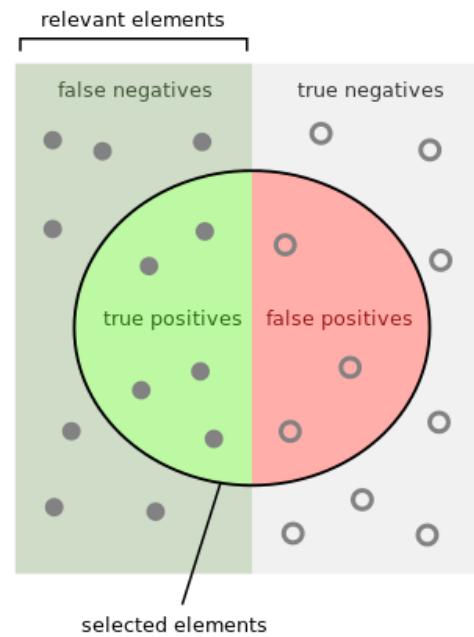
Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)
- Accuracy = $(TP + TN) / \text{total}$
- Misclassification = $(FP + FN) / \text{total}$

Precision, Recall, F1

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	



- Precision = correctly predict = $TP / (TP + FP)$
- Recall = coverage = $TP / (TP + FN)$
- F1 = $(2 * \text{pre} * \text{rec}) / (\text{pre} + \text{rec})$

How many selected items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{red} + \text{green}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{green}}{\text{green} + \text{white}}$$



Precision, Recall, F1: Average

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- Precision = correctly predict = $TP / (TP + FP)$
- Recall = coverage = $TP / (TP + FN)$
- F1 = $(2 * \text{pre} * \text{rec}) / (\text{pre} + \text{rec})$

- What is a positive class?
 - 1) Direct target marketing? Yes
 - 2) Intelligent diagnosis to predict “Corona” or “Flu” – **both** are important.
- If there is no positive (all classes are important)
 - Macro Average
 - Weighted Average (Micro Average) by the amount of data in that class



Please [cite us](#) if you use the software.

[sklearn.metrics.confusion_matrix](#)

Examples using

[sklearn.metrics.confusion_matrix](#)

.

Read more in the [User Guide](#).

Parameters: **y_true** : array-like of shape (n_samples),

Ground truth (correct) target values.

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

of shape (n_samples,

s returned by a classifier.

shape (n_classes), default=None

ex the matrix. This may be used to reorder or select a subset of labels. If None is given, those once in y_true or y_pred are used in sorted order.

array-like of shape (n_samples,), default=None

normalize : {'true', 'pred', 'all'}, default=None

Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population. If None, confusion matrix will not be normalized.

sklearn.metrics.classification_report

```
sklearn.metrics.classification_report(y_true, y_pred, labels=None, target_names=None, sample_weight=None, digits=2, output_dict=False, zero_division='warn')
```

[\[source\]](#)

Build a text report showing the main classification metrics

Read more in the [User Guide](#).

Parameters:**y_true : 1d array-like, or label indicator array**

Ground truth (correct) target values.

y_pred : 1d array-like, or label indicator array

Estimated targets as returned by a classifier.

labels : array, shape = [n_labels]

Optional list of label indices to include.

target_names : list of strings

Optional display names matching classes.

sample_weight : array-like of shape [n_samples]

Sample weights.

digits : int

Number of digits for formatting output float values. The returned values will not be rounded.

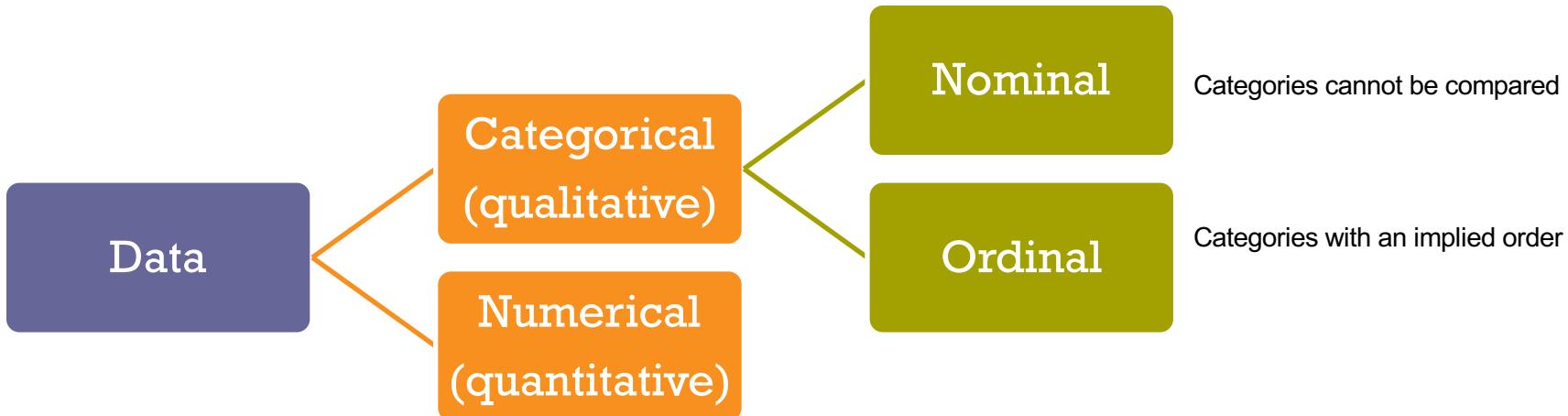
Examples

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support
  class 0       0.50      1.00      0.67      1
  class 1       0.00      0.00      0.00      1
  class 2       1.00      0.67      0.80      3
                                              accuracy         0.60      5
                                              macro avg       0.50      0.56      0.49      5
                                              weighted avg    0.70      0.60      0.61      5
<BLANKLINE>
accuracy           0.60      5
macro avg          0.50      0.56      0.49      5
weighted avg        0.70      0.60      0.61      5
<BLANKLINE>
>>> y_pred = [1, 1, 0]
>>> y_true = [1, 1, 1]
>>> print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
              precision    recall  f1-score   support
  1       1.00      0.67      0.80      3
  2       0.00      0.00      0.00      0
  3       0.00      0.00      0.00      0
                                              accuracy         0.80      3
                                              macro avg       0.67      0.67      0.67      3
                                              weighted avg    0.80      0.67      0.67      3
<BLANKLINE>
```

+

Non-numeric variables (Model M2)

Terminology: Kinds of data (recap)



Ordinal: Recode

Grade	GradeNum
A	4.00
B+	3.50
B	3.00
C+	2.50
C	2.00
D+	1.50
D	1.00
F	0.00

Size	SizeNum
XL	5
L	4
M	3
S	2
XS	1



Categorical

- Dummy coding = (n-1) dummy codes

Branch	BranchNum	D_BKK	B_Patum	D_Non	D_BKK	B_Patum
BKK	1	1	0	0	1	0
Patumtani	2	0	1	0	0	1
Nontaburi	3	0	0	1	0	0 reference

Search the docs ...

Input/output

General functions

Series

DataFrame

Pandas arrays

Panel

Index objects

Date offsets

Window

GroupBy

Resampling

Style

pandas.get_dummies

`pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None) → 'DataFrame'` [\[source\]](#)

Convert categorical variable into dummy/indicator variables.

Parameters: `data : array-like, Series, or DataFrame`

Data of which to get dummy indicators.

`prefix : str, list of str, or dict of str, default None`

String to append DataFrame column names. Pass a list with length equal to the number of columns when calling `get_dummies` on a DataFrame. Alternatively, `prefix` can be a dictionary mapping column names to prefixes.

`prefix_sep : str, default '_'`

If appending prefix, separator/delimiter to use. Or pass a list or dictionary as with `prefix`.

`dummy_na : bool, default False`

Add a column to indicate NaNs, if False NaNs are ignored.

```
>>> pd.get_dummies(pd.Series(list('abcaa')), drop_first=True)
   b   c
0  0   0
1  1   0
2  0   1
3  0   0
4  0   0
```

`like, default None`

Passes in the DataFrame to be encoded. If `columns` is None then all the columns with `object` or `category` converted.

`default False`

Dummy-encoded columns should be backed by a `SparseArray` (True) or a regular NumPy array (False).