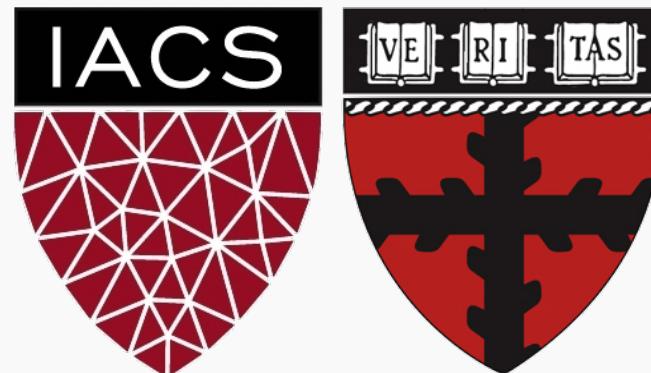
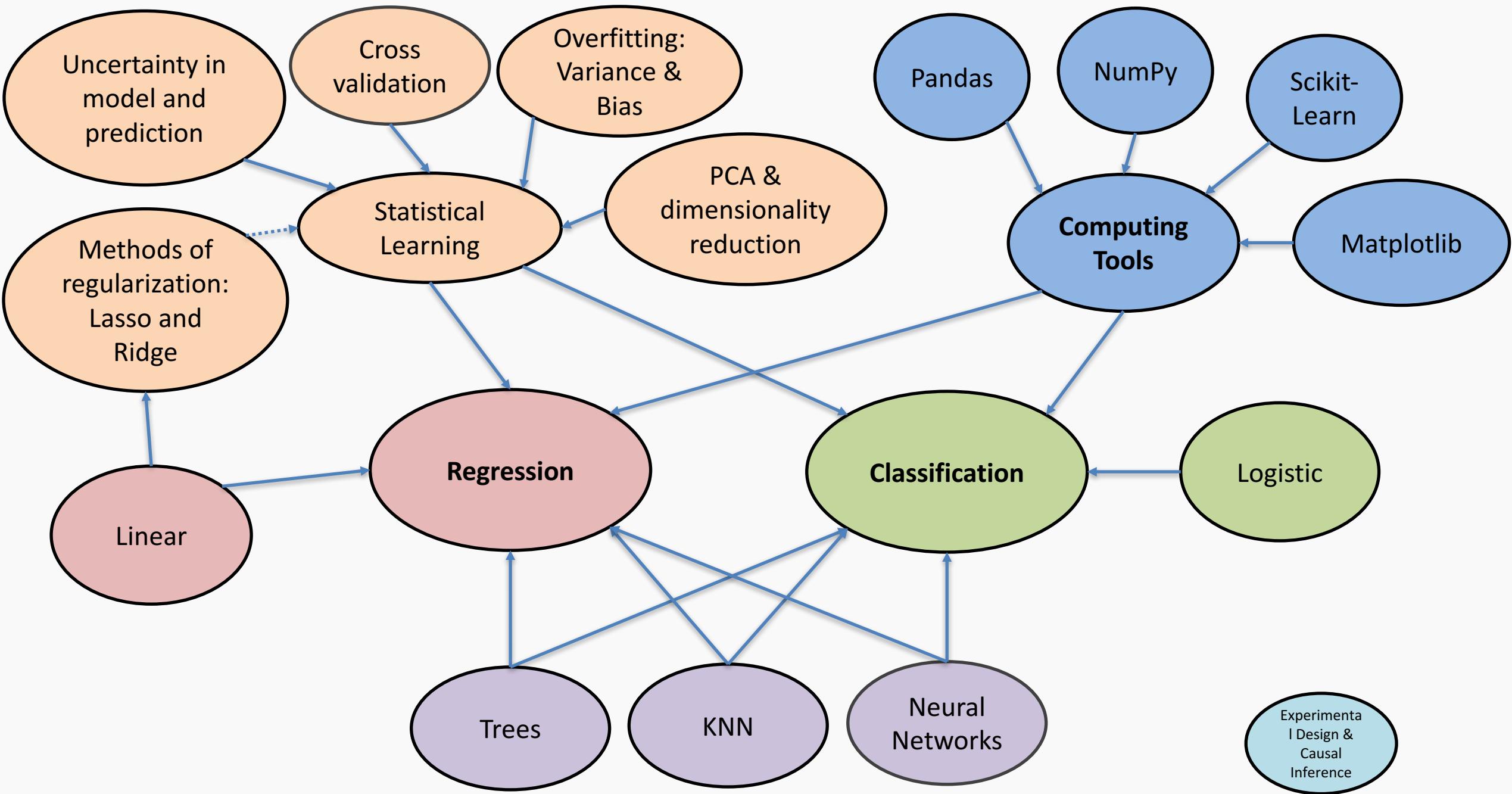
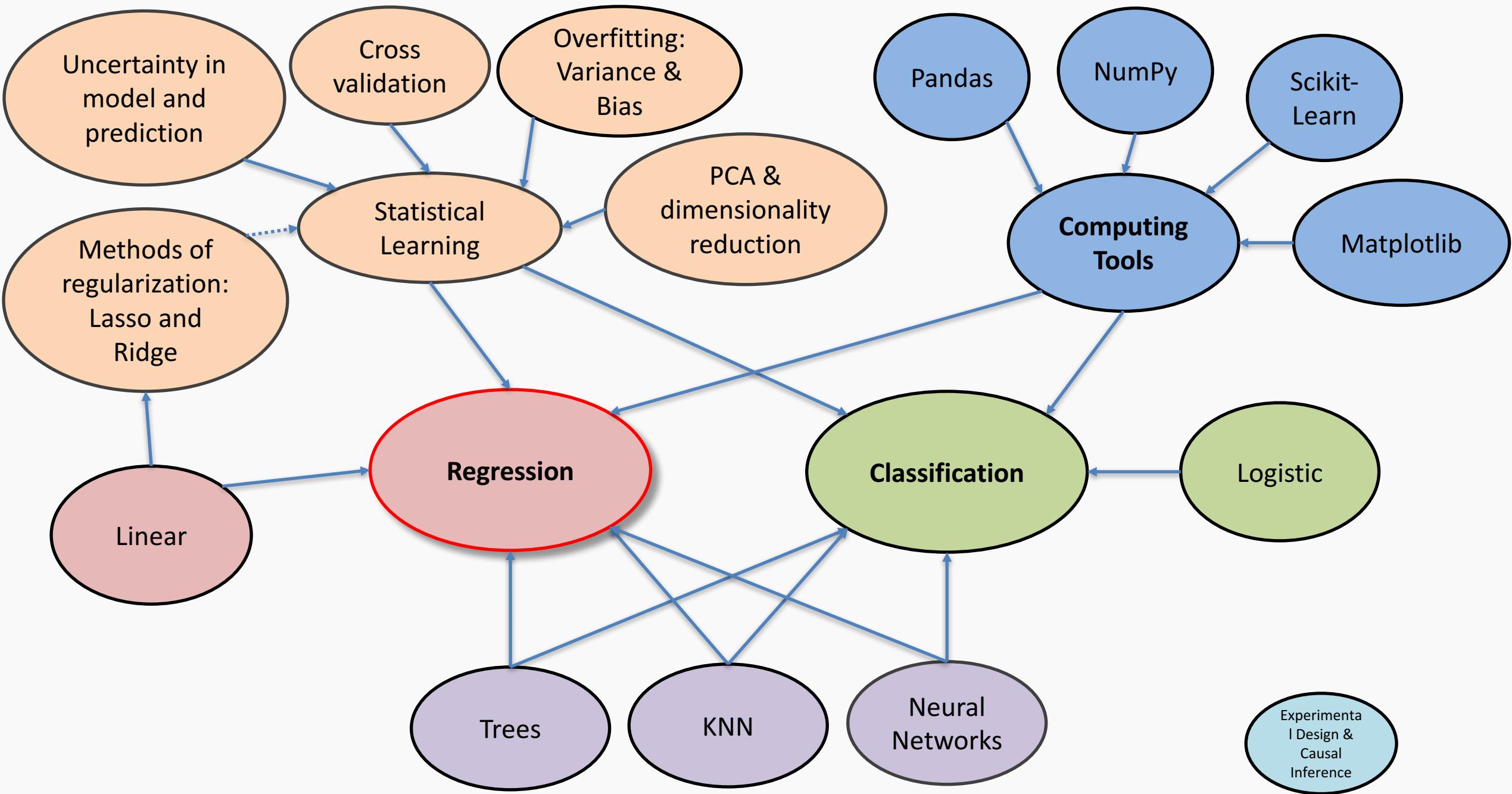


Lecture 24: Review

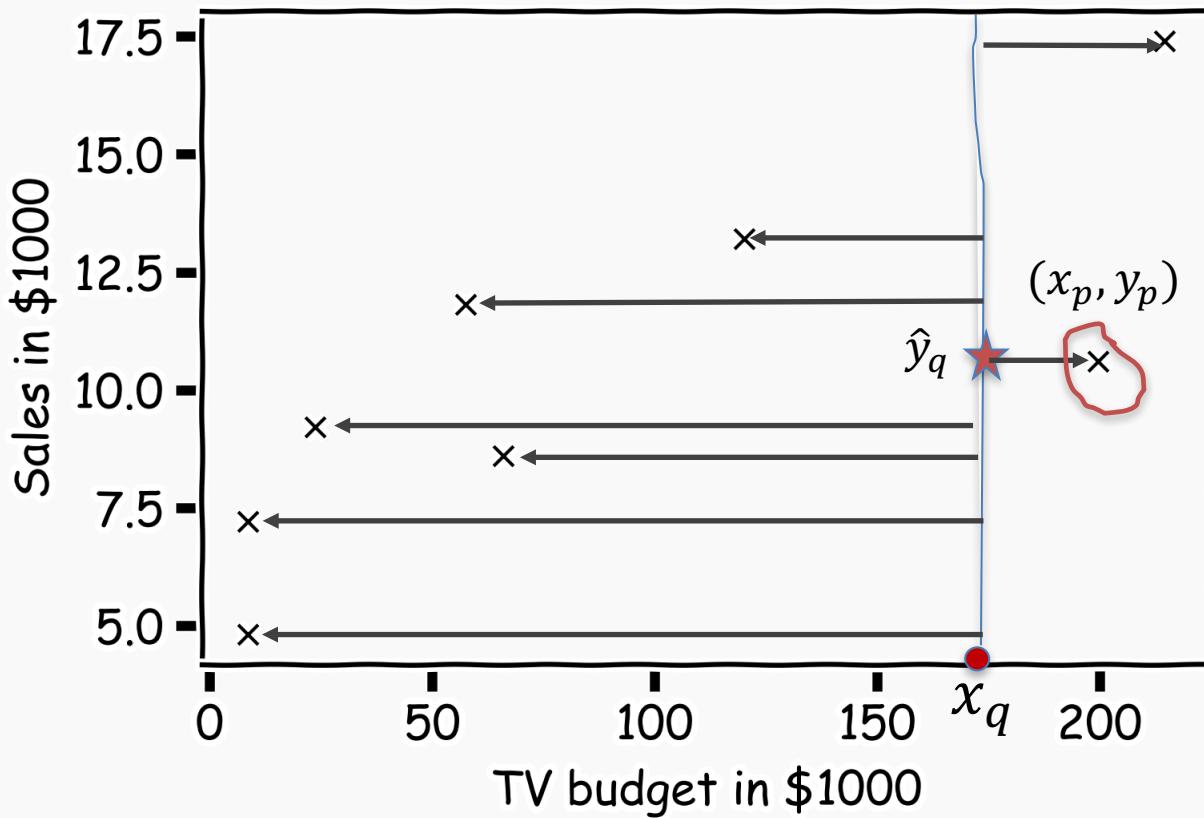
CS109A Introduction to Data Science
Pavlos Protopapas, Kevin Rader and Chris Tanner







Simple Prediction Model



What is \hat{y}_q at some x_q ?

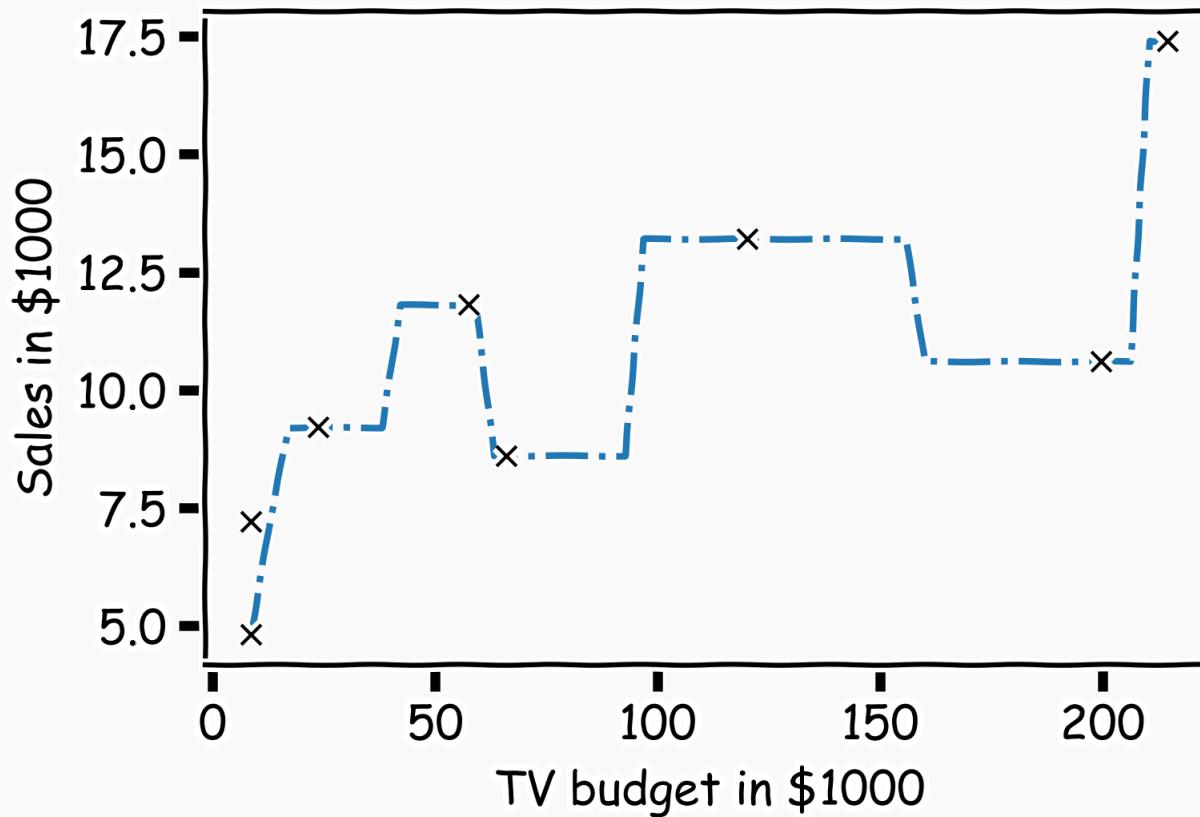
Find distances to all other points $D(x_q, x_i)$

Find the nearest neighbor, (x_p, y_p)

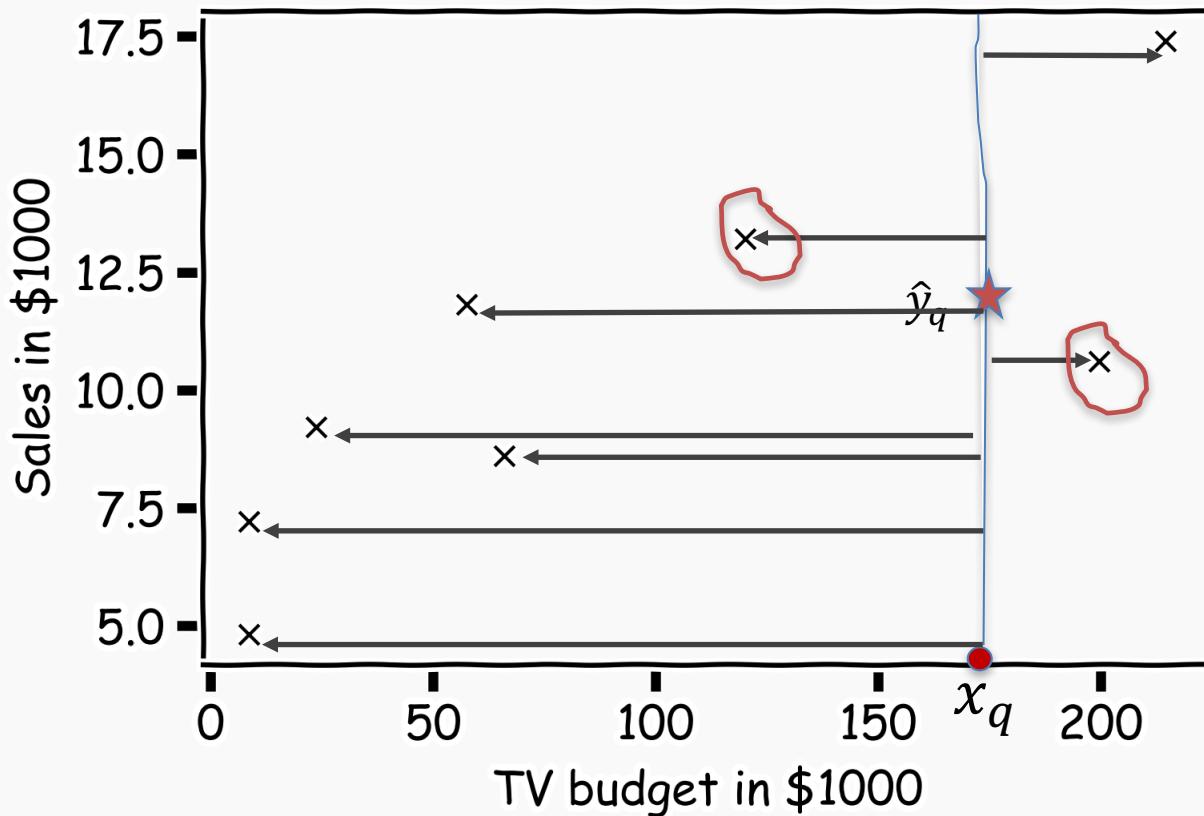
Predict $\hat{y}_q = y_p$

Simple Prediction Model

Do the same for “all” x 's



Extend the Prediction Model



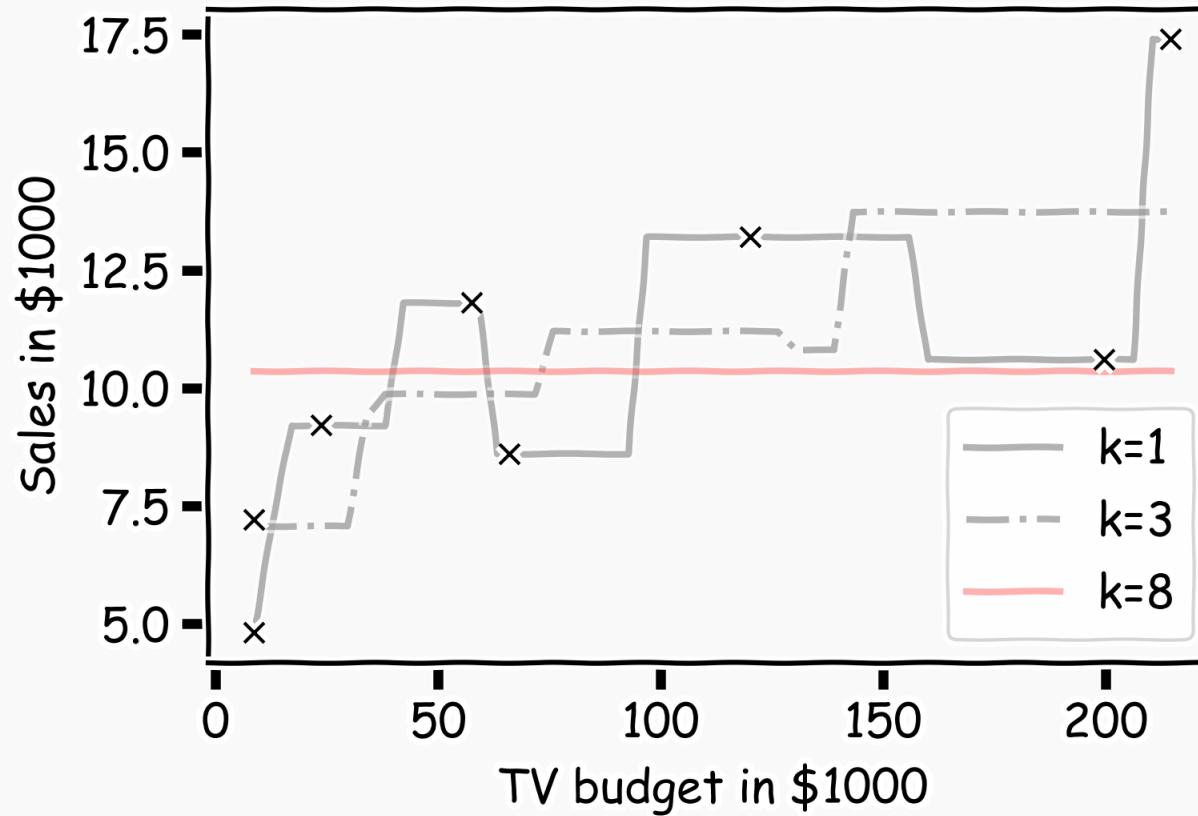
What is \hat{y}_q at some x_q ?

Find distances to all other points $D(x_q, x_i)$

Find the k-nearest neighbors, x_{q_1}, \dots, x_{q_k}

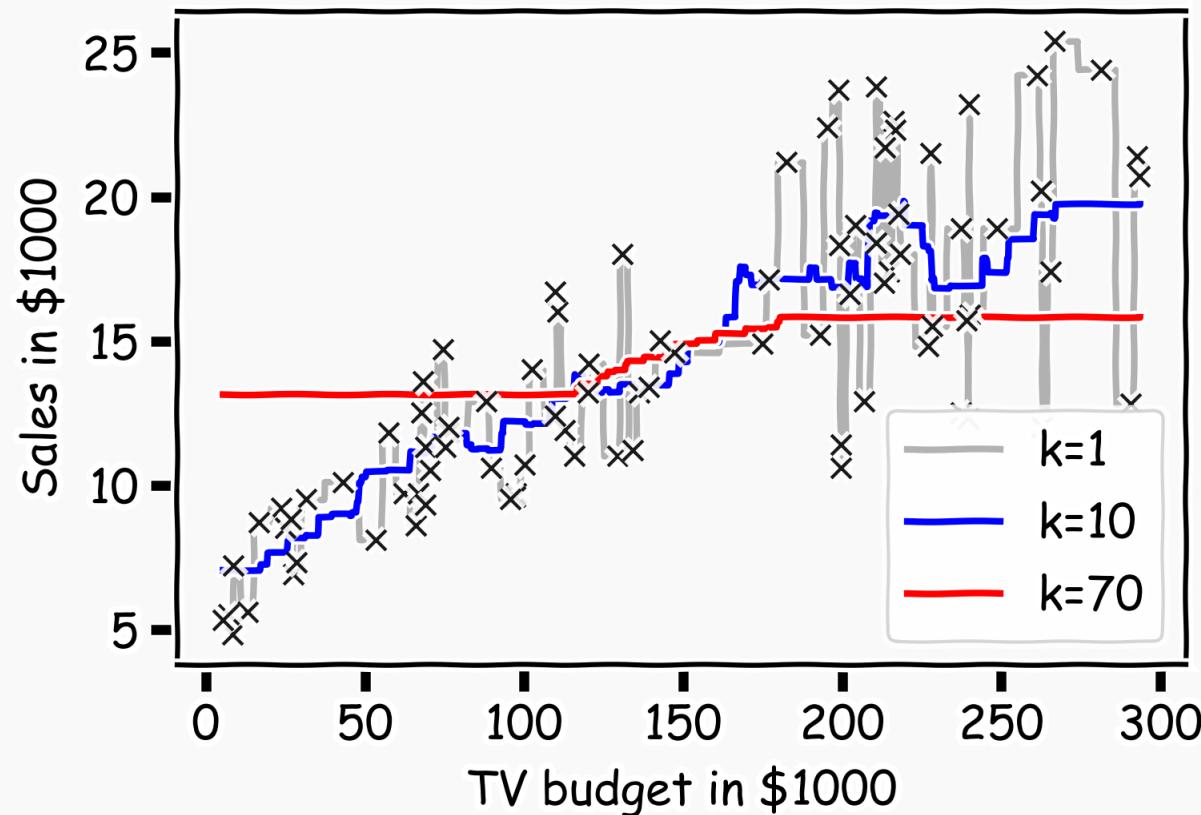
Predict $\hat{y}_q = \frac{1}{k} \sum_i^k y_{q_i}$

Simple Prediction Models



Simple Prediction Models

We can try different k-models on more data



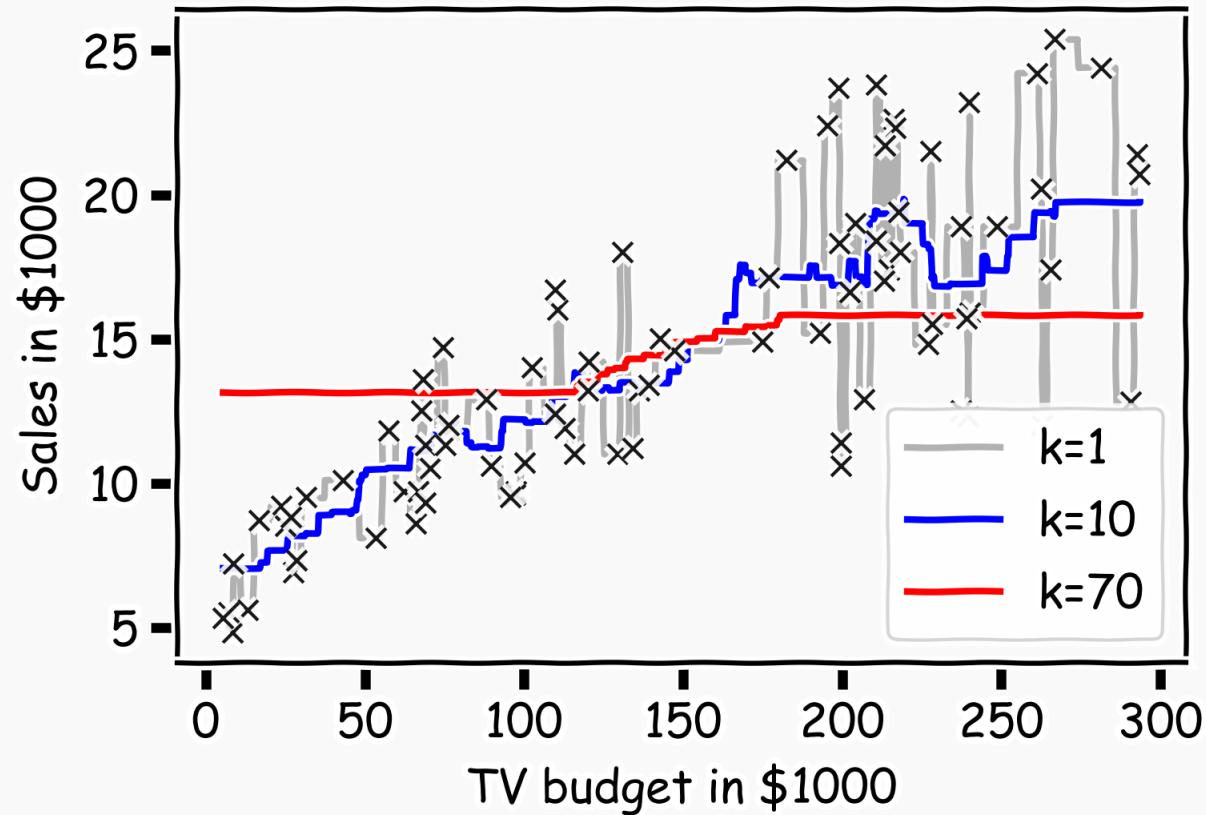
k-Nearest Neighbors - kNN

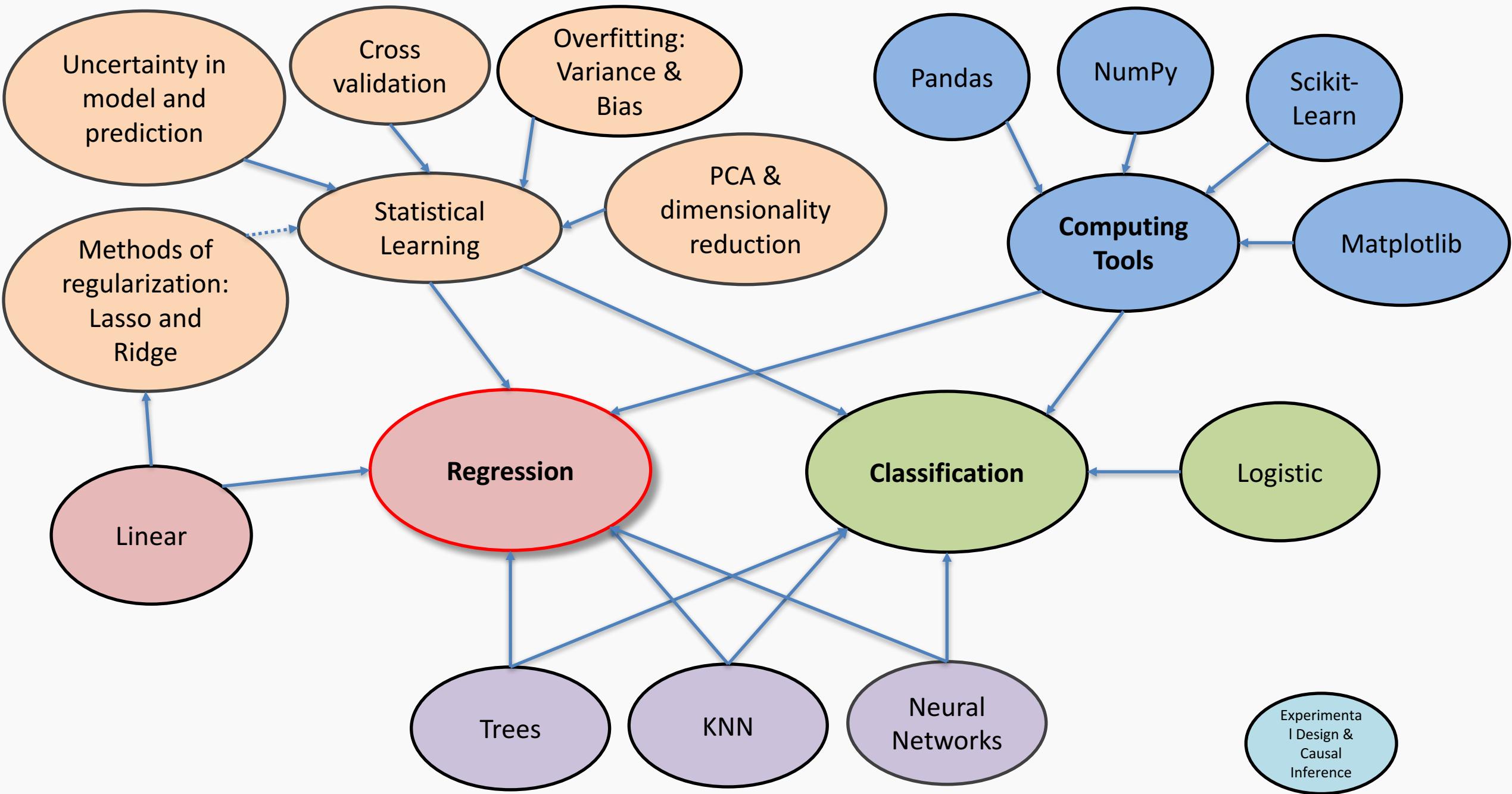
For a fixed a value of k , the predicted response for the i -th observation is the average of the observed response of the k -closest observations:

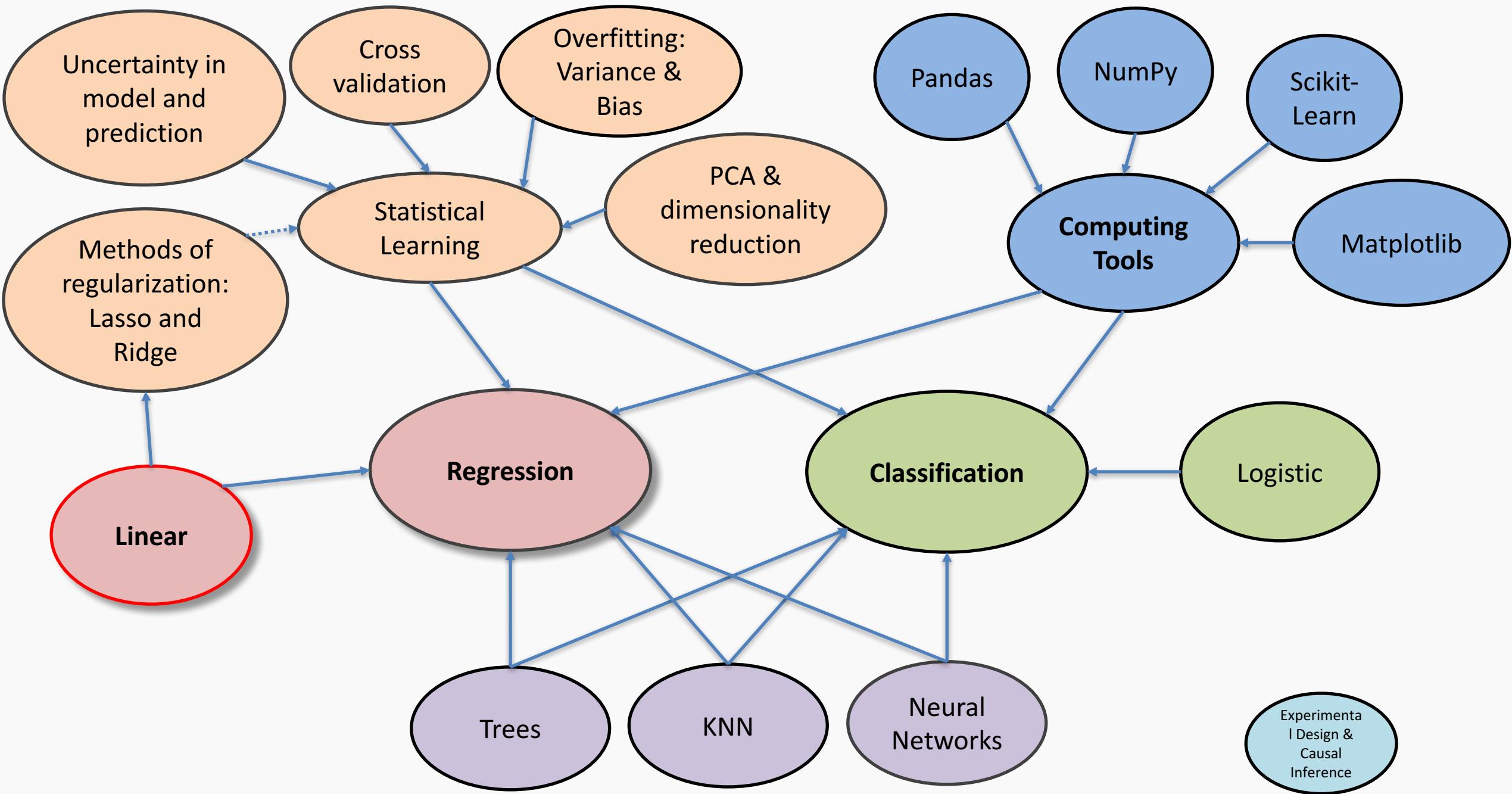
$$\hat{y}_n = \frac{1}{k} \sum_{i=1}^k y_{n_i}$$

where $\{x_{n_1}, \dots, x_{n_k}\}$ are the k observations most similar to x_i (*similar* refers to a notion of distance between predictors).

ED quiz: Lecture 4 | part 1

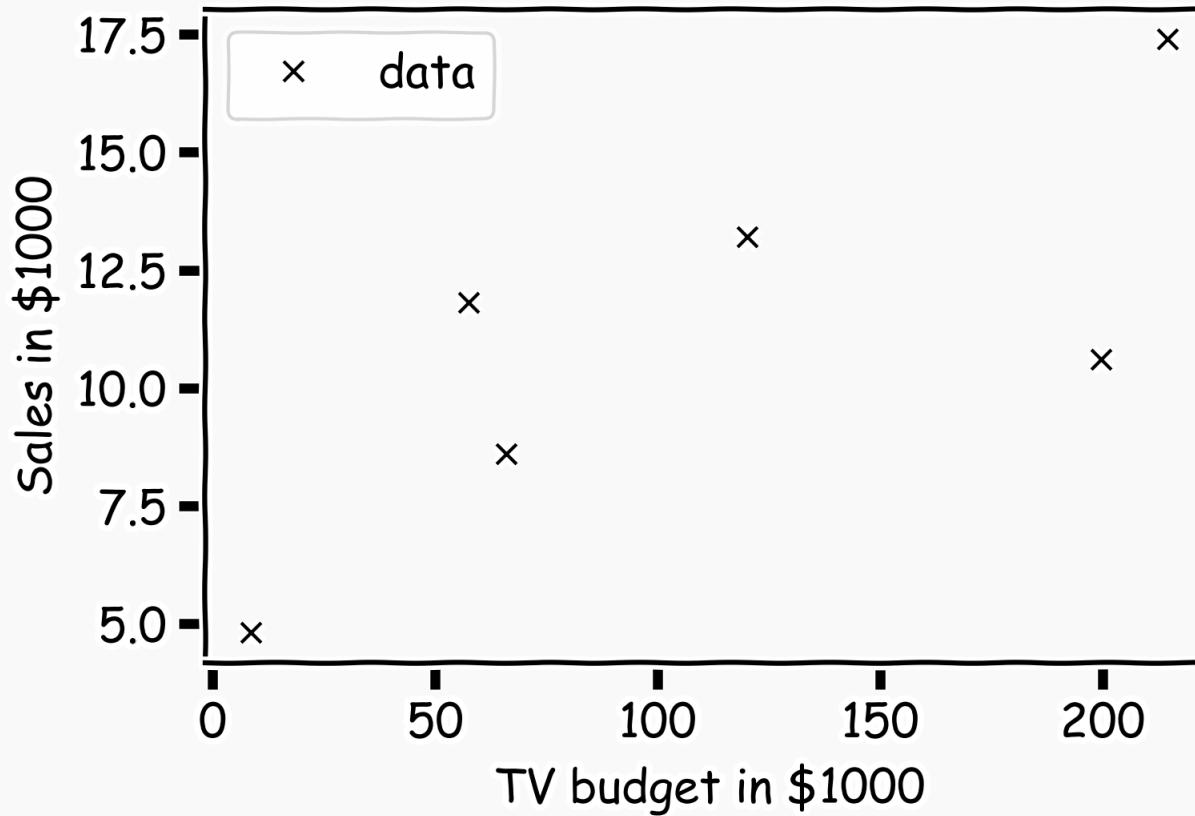






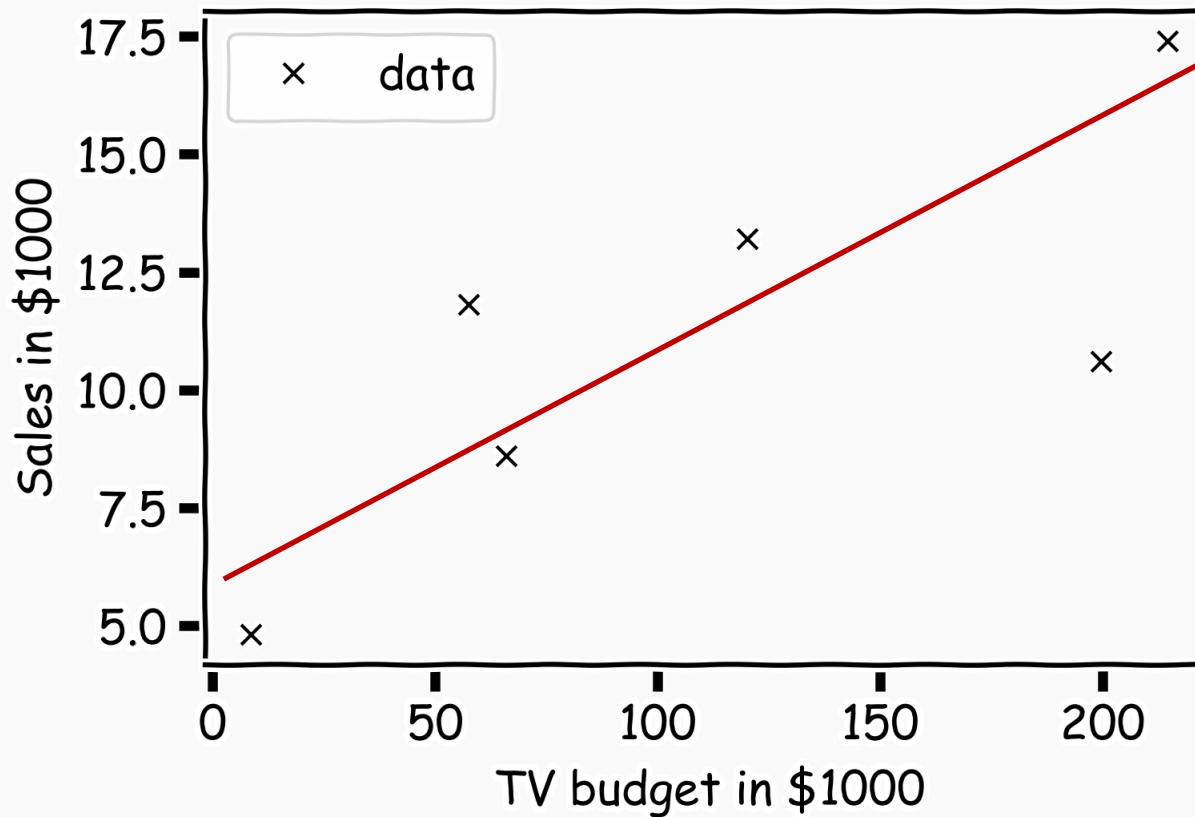
Estimate of the regression coefficients

For a given data set



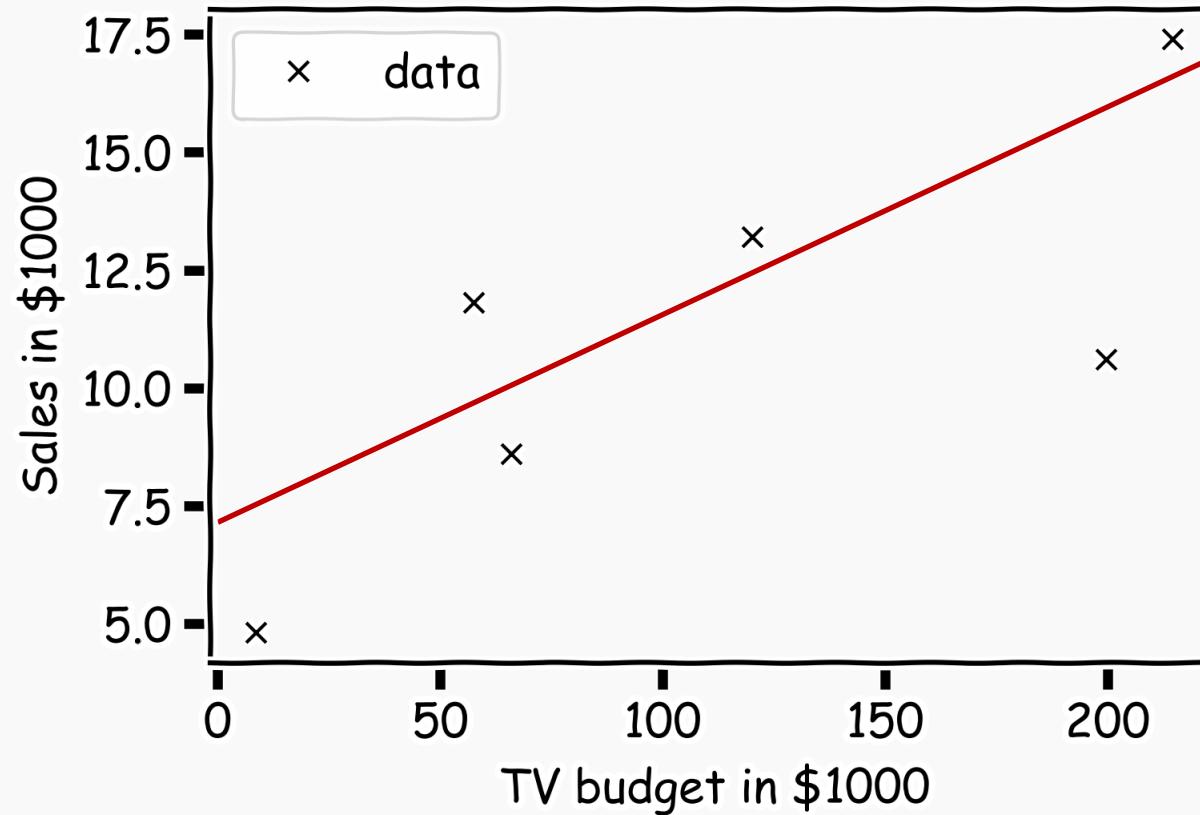
Estimate of the regression coefficients (cont)

Is this line good?



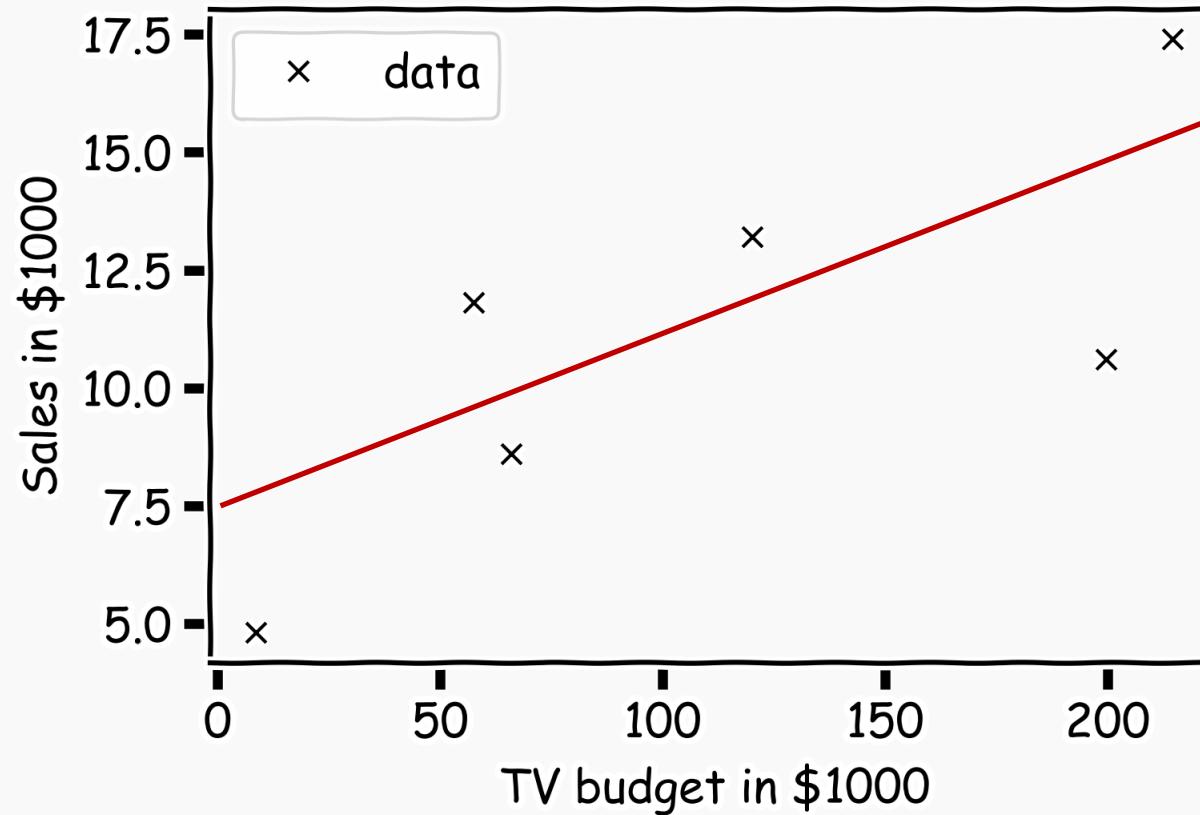
Estimate of the regression coefficients (cont)

Maybe this one?



Estimate of the regression coefficients (cont)

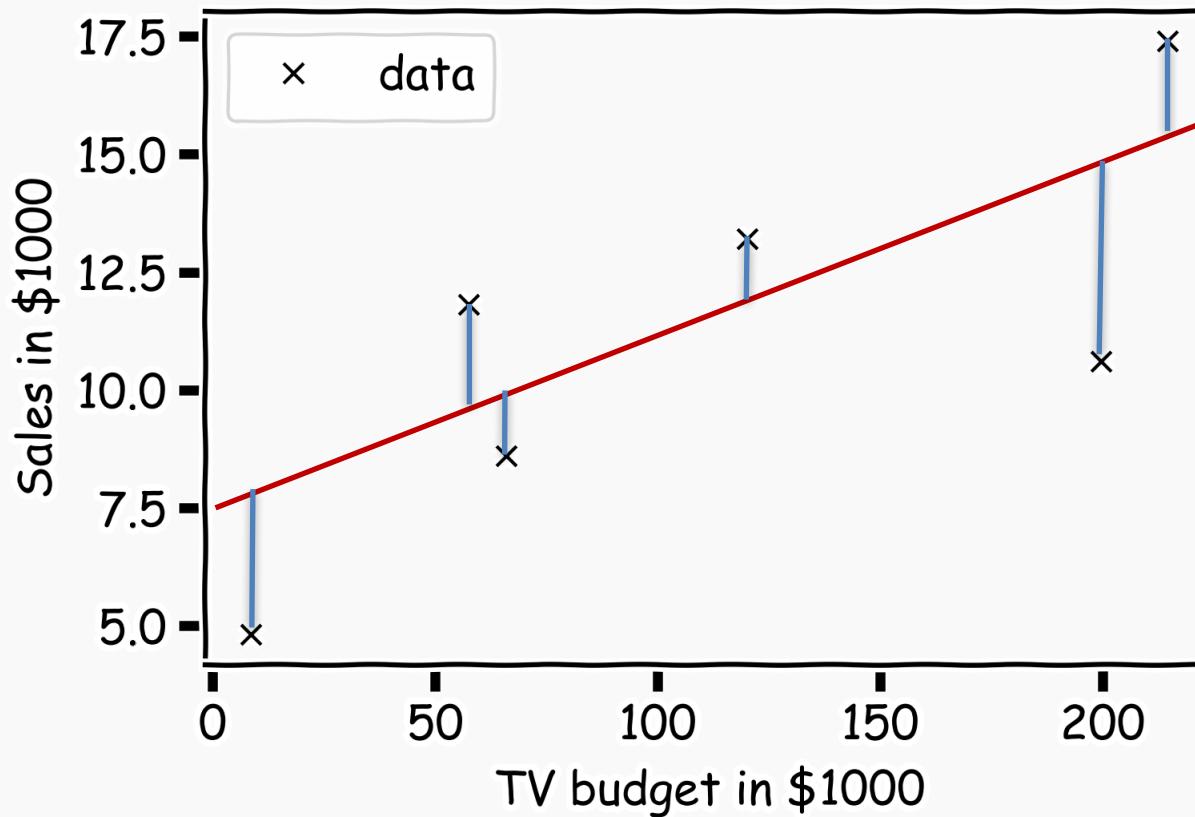
Or this one?



Estimate of the regression coefficients (cont)

Question: Which line is the best?

First calculate the residuals



Estimate of the regression coefficients (cont)

Again we use MSE as our **loss function**,

$$L(\beta_0, \beta_1) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n [y_i - (\beta_1 X + \beta_0)]^2.$$

We choose $\hat{\beta}_1$ and $\hat{\beta}_0$ in order to minimize the predictive errors made by our model, i.e. minimize our loss function.

Then the optimal values for $\hat{\beta}_0$ and $\hat{\beta}_1$ should be:

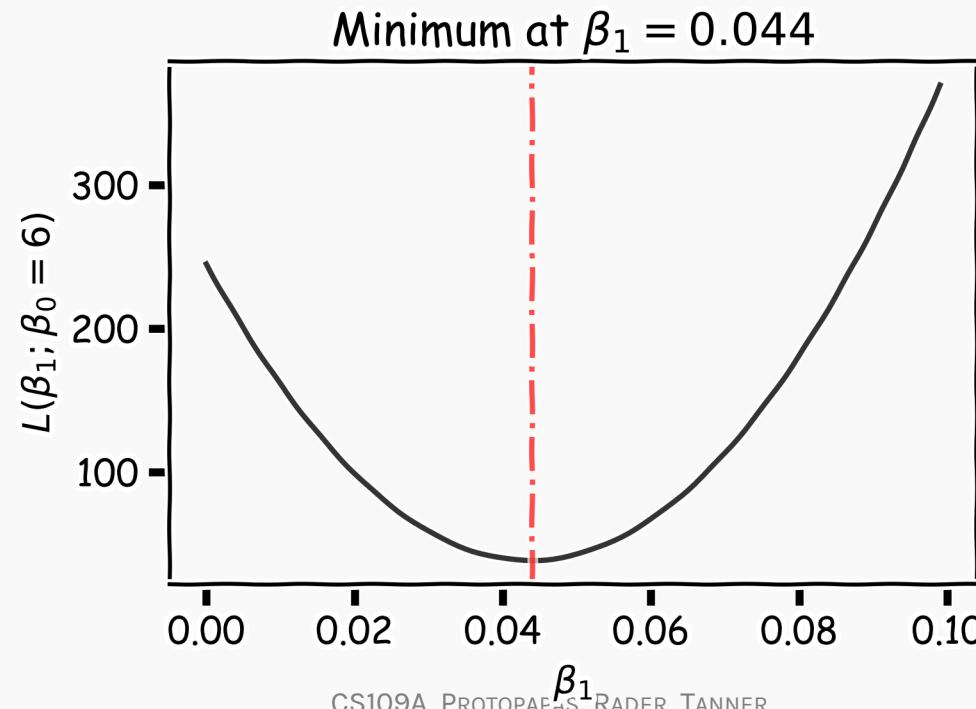
$$\hat{\beta}_0, \hat{\beta}_1 = \operatorname{argmin}_{\beta_0, \beta_1} L(\beta_0, \beta_1).$$



Estimate of the regression coefficients: brute force

A way to estimate $\operatorname{argmin}_{\beta_0, \beta_1} L$ is to calculate the loss function for every possible β_0 and β_1 . Then select the β_0 and β_1 where the loss function is minimum.

E.g. the loss function for different β_1 when β_0 is fixed to be 6:



Estimate of the regression coefficients: exact method

Take the partial derivatives of L with respect to β_0 and β_1 , set to zero, and find the solution to that equation. This procedure will give us explicit formulae for $\hat{\beta}_0$ and $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

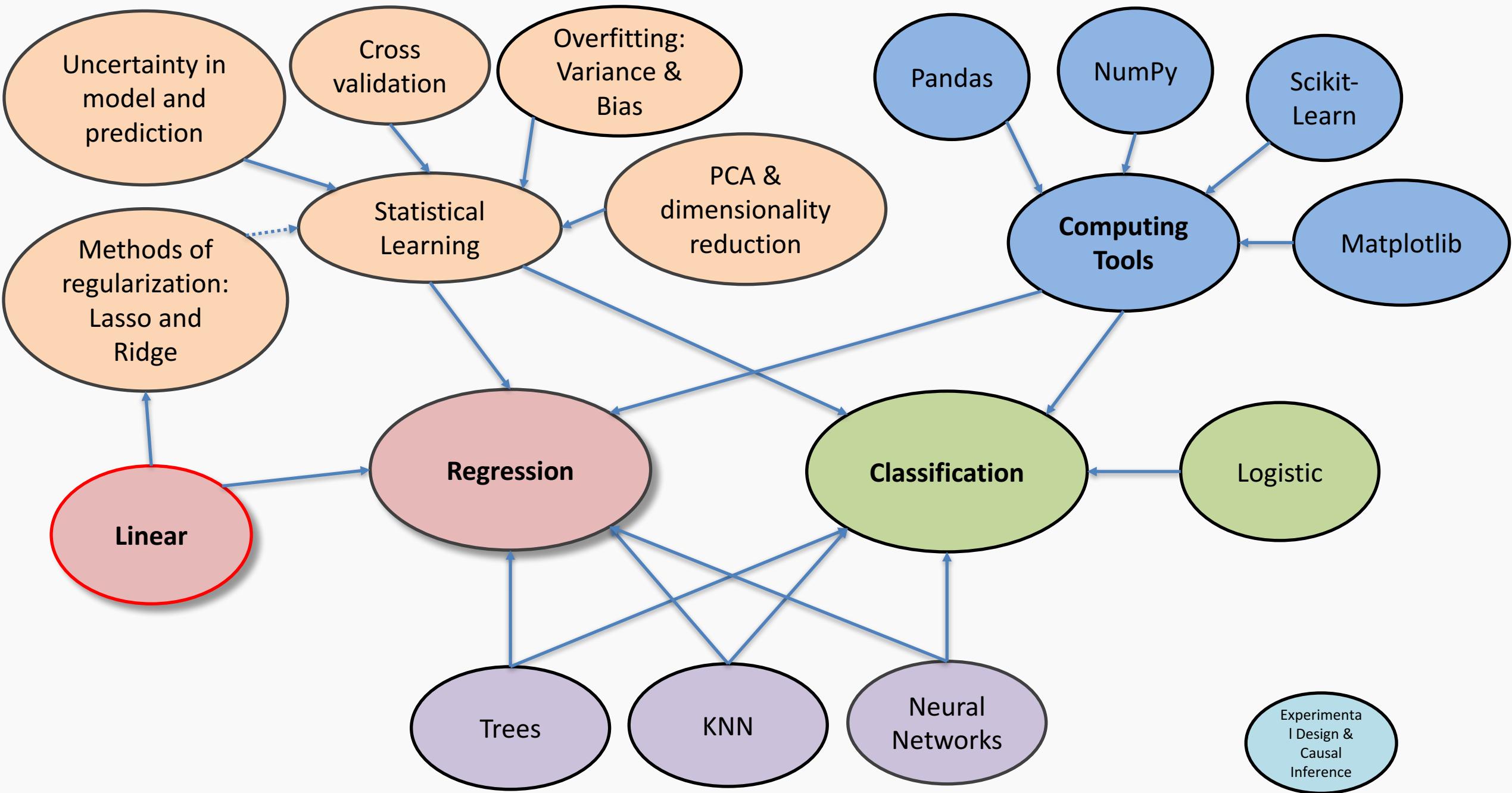
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

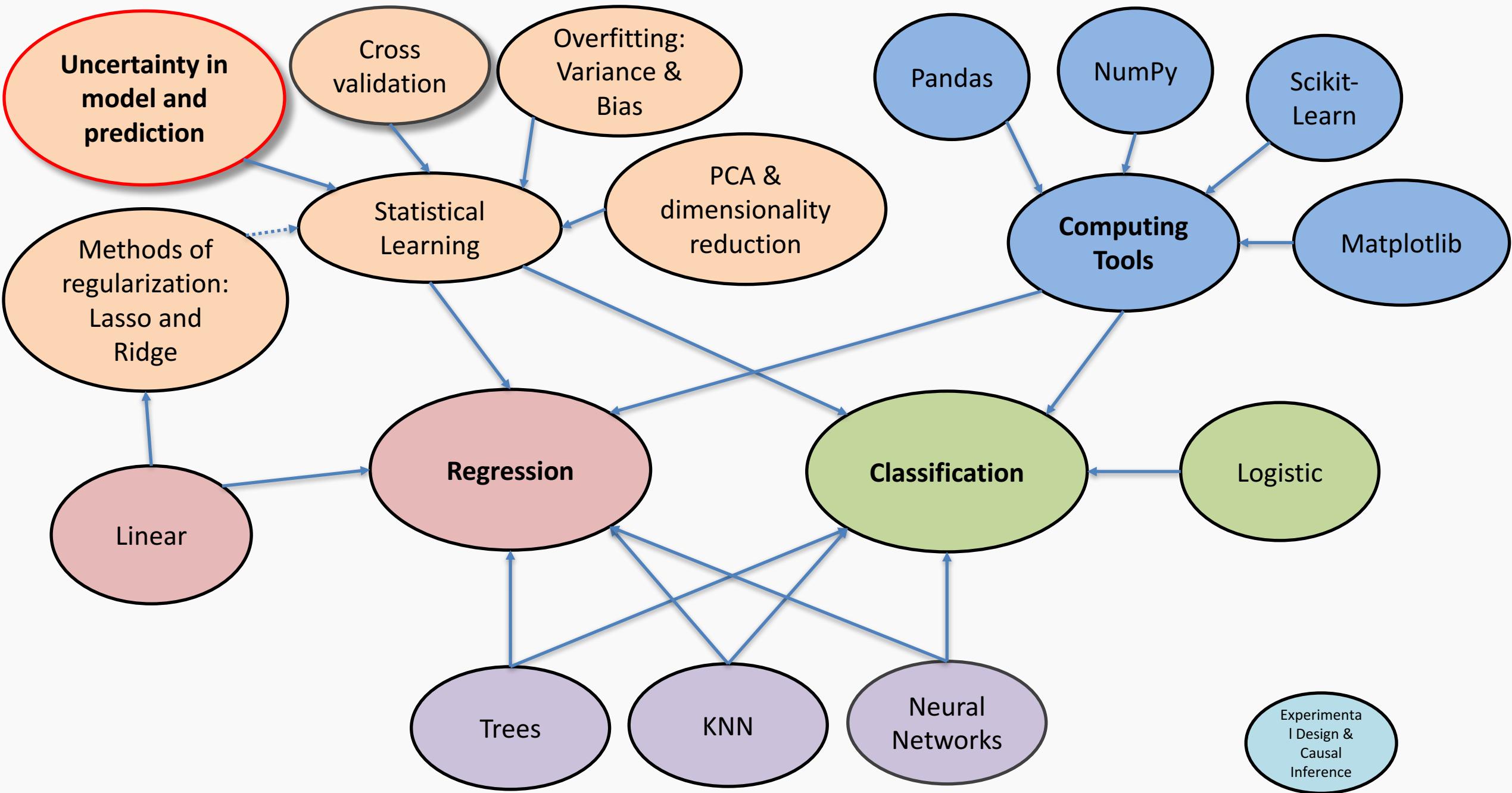
where \bar{y} and \bar{x} are sample means.

The line:

$$\hat{Y} = \hat{\beta}_1 X + \hat{\beta}_0$$

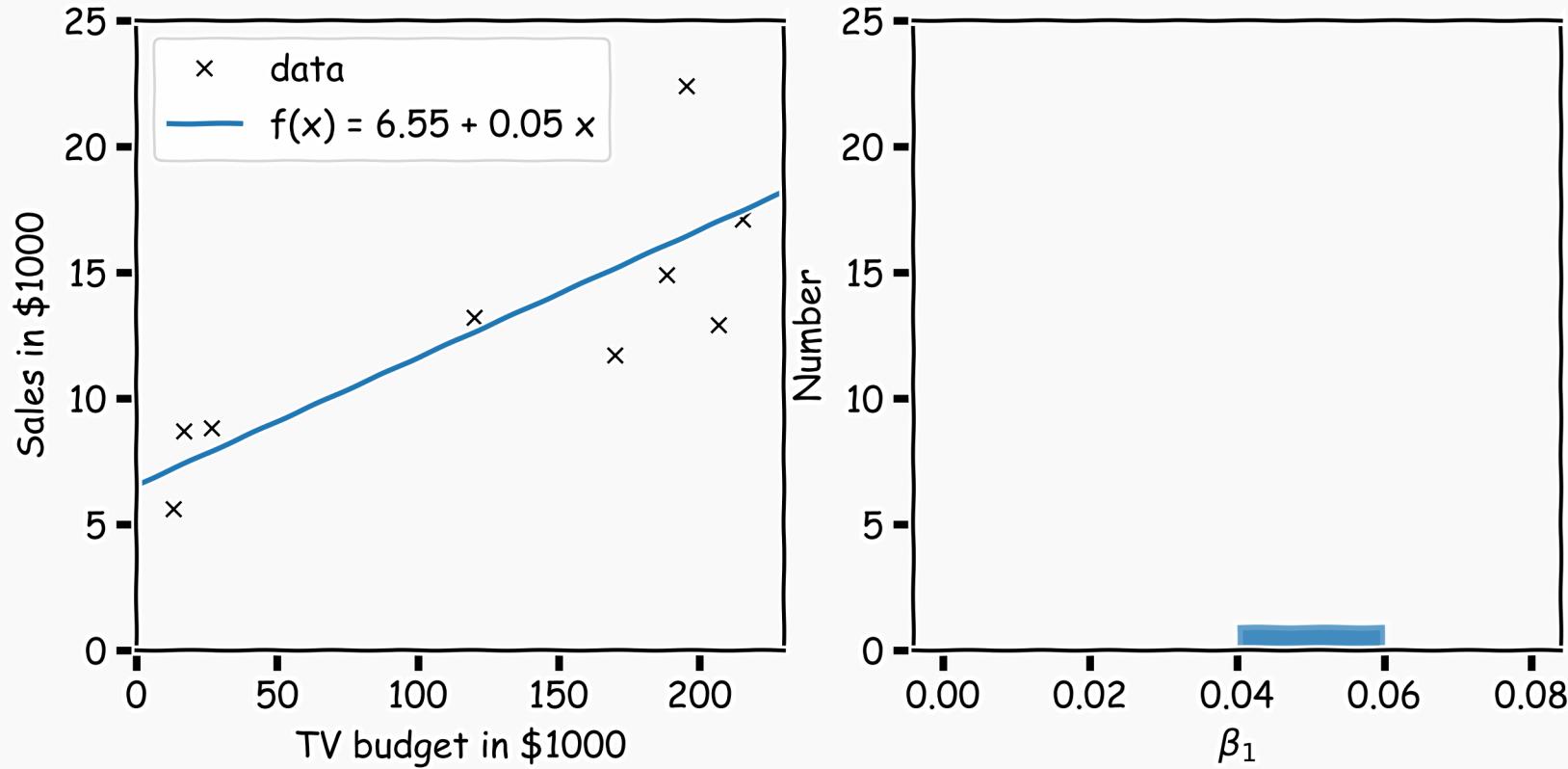
is called the **regression line**.





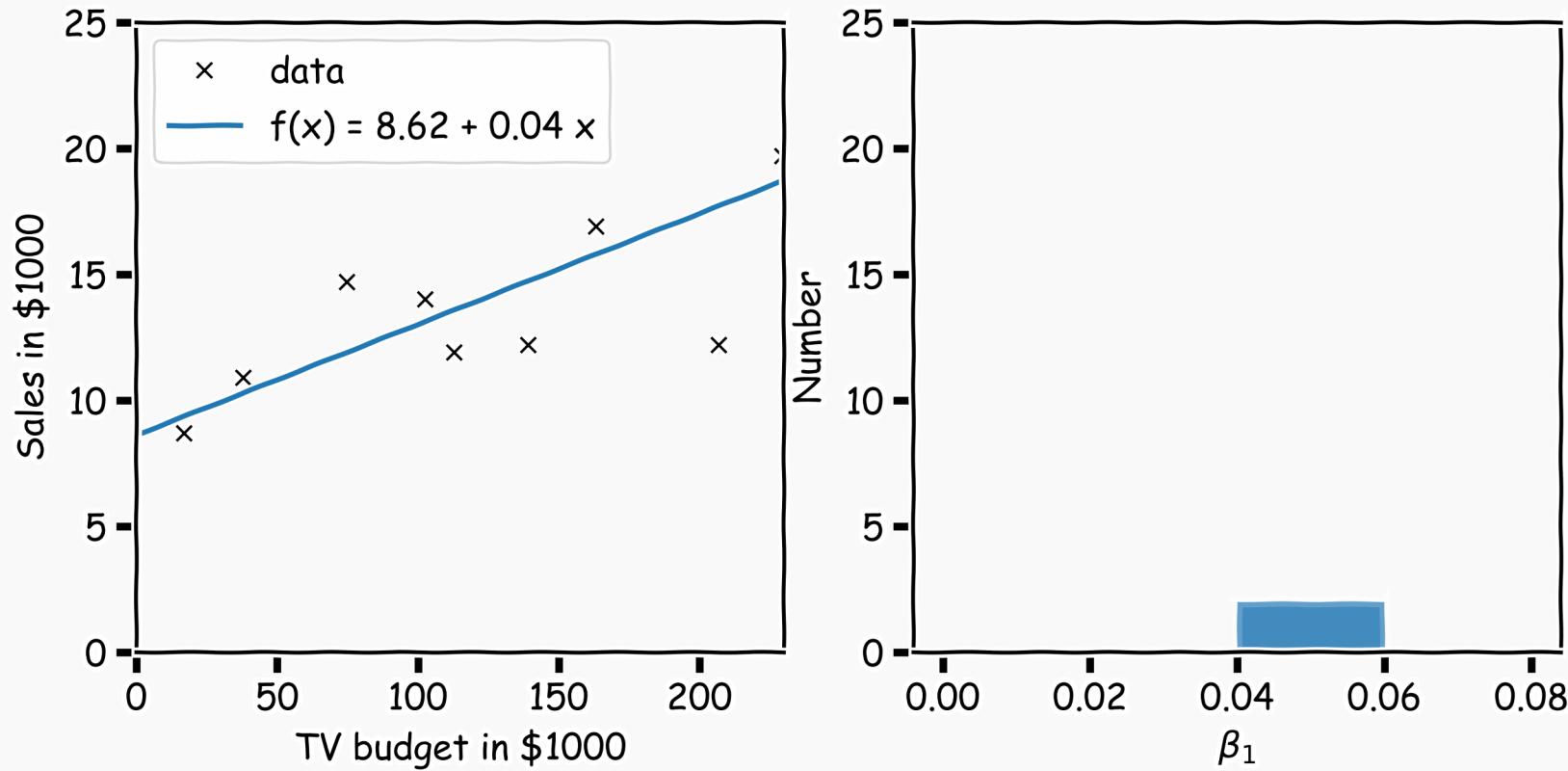
Confidence intervals for the predictors estimates (cont)

In our magical realisms, we can now sample multiple times



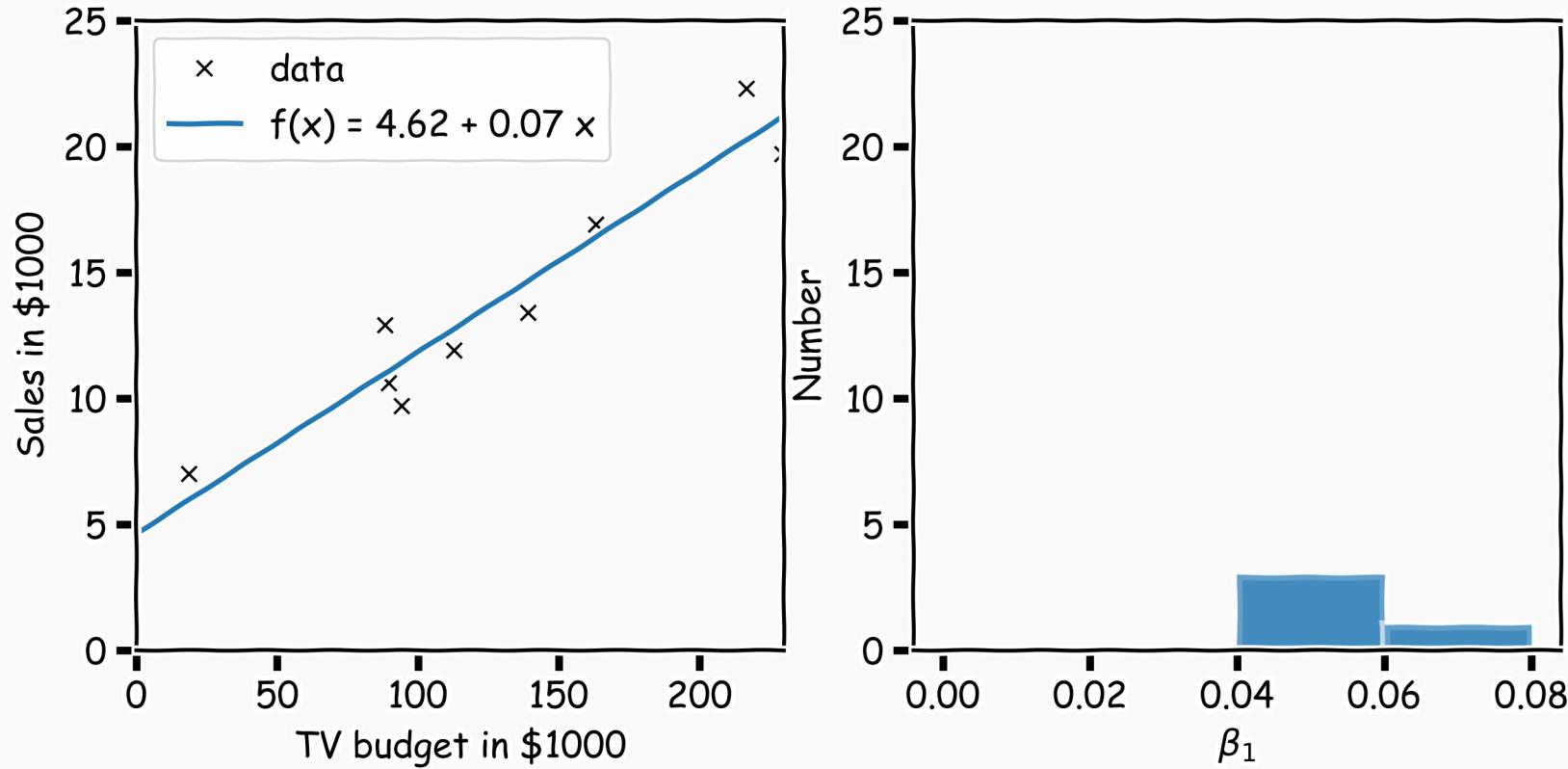
Confidence intervals for the predictors estimates (cont)

Another sample



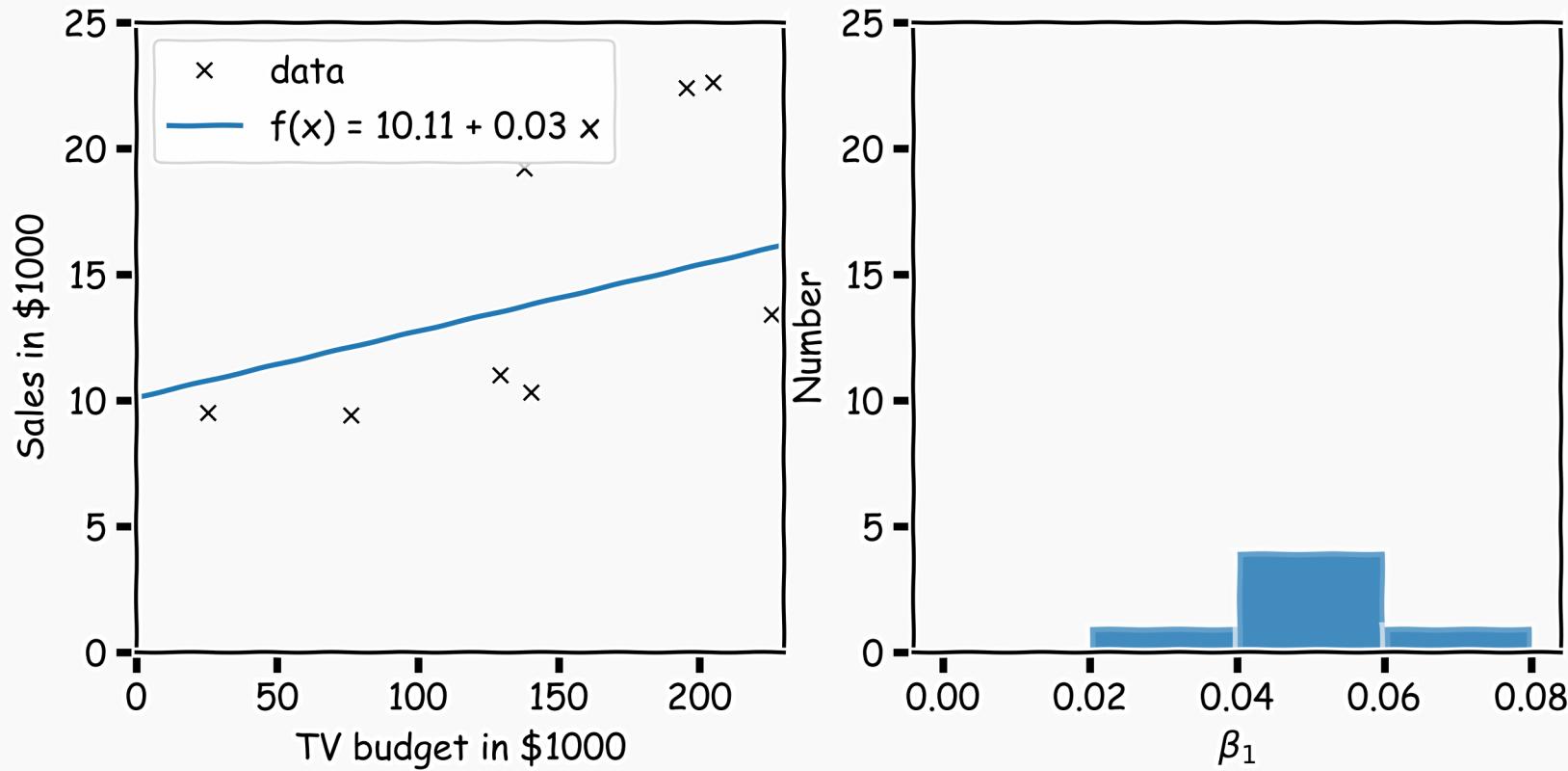
Confidence intervals for the predictors estimates (cont)

Another sample



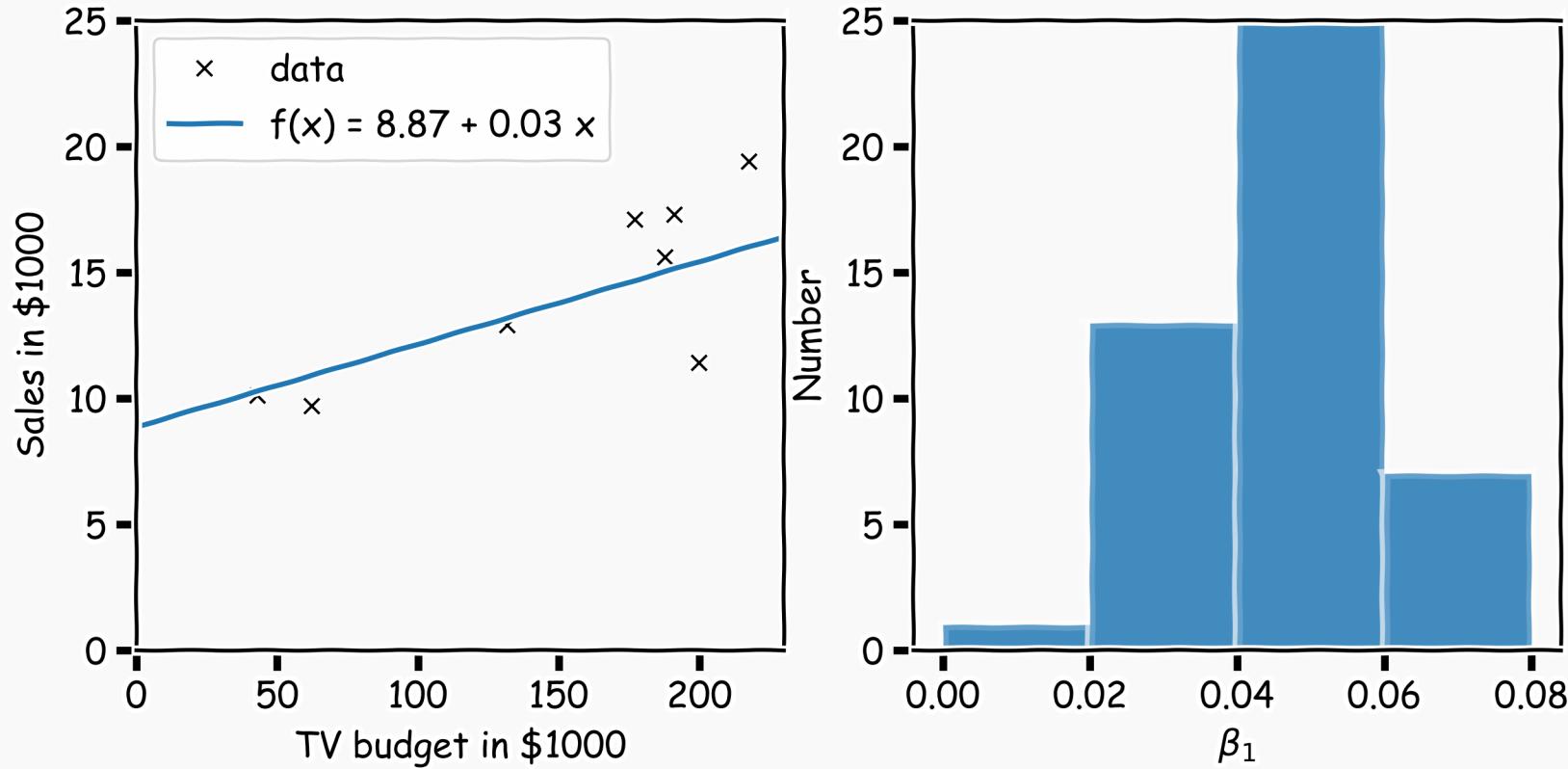
Confidence intervals for the predictors estimates (cont)

And another sample



Confidence intervals for the predictors estimates (cont)

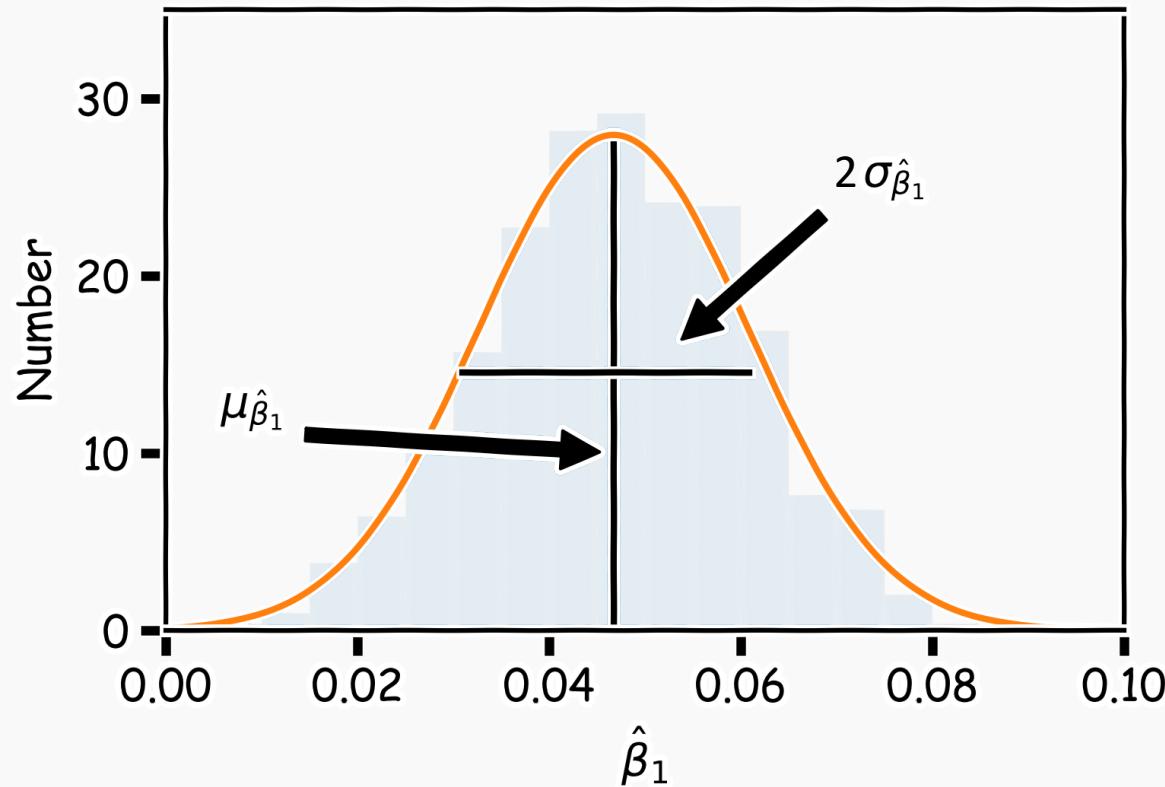
Repeat this for 100 times



Confidence intervals for the predictors estimates (cont)

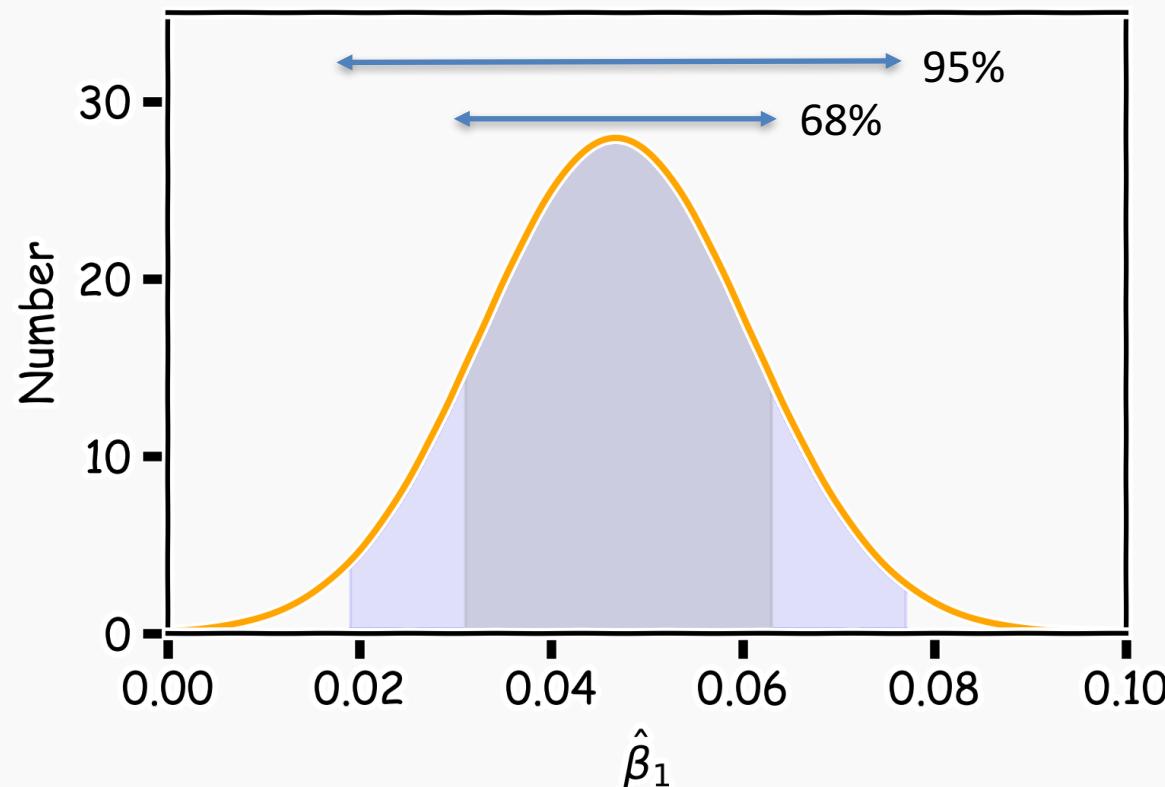
We can now estimate the mean and standard deviation of all the estimates $\hat{\beta}_1$.

The variance of $\hat{\beta}_0$ and $\hat{\beta}_1$ are also called their **standard errors**, $SE(\hat{\beta}_0), SE(\hat{\beta}_1)$.



Confidence intervals for the predictors estimates (cont)

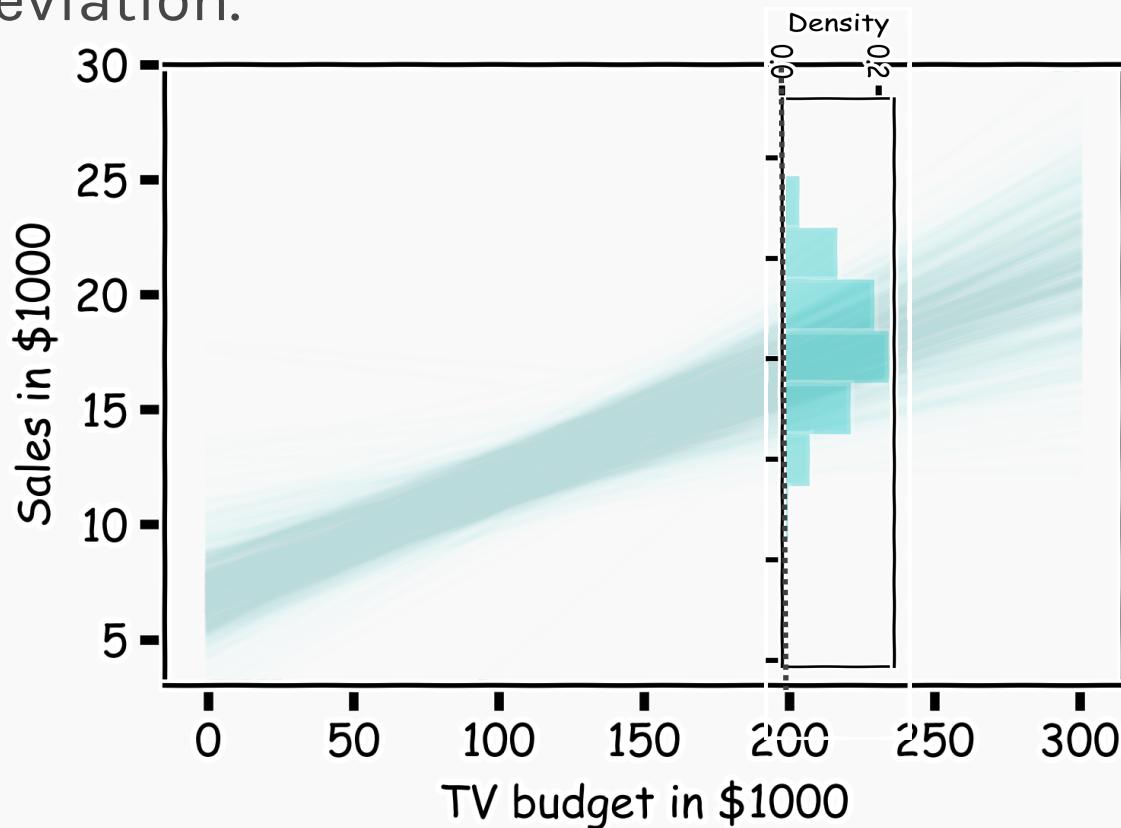
Finally we can calculate the confidence intervals, which are the ranges of values such that the **true** value of β_1 is contained in this interval with n percent probability.



How well do we know \hat{f} ?

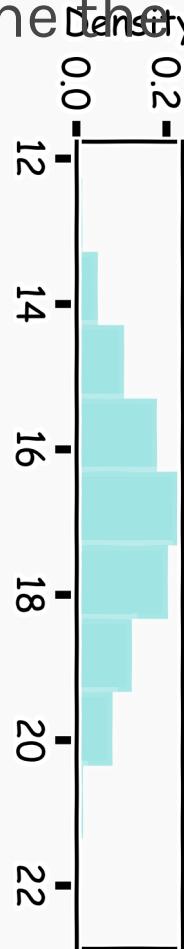
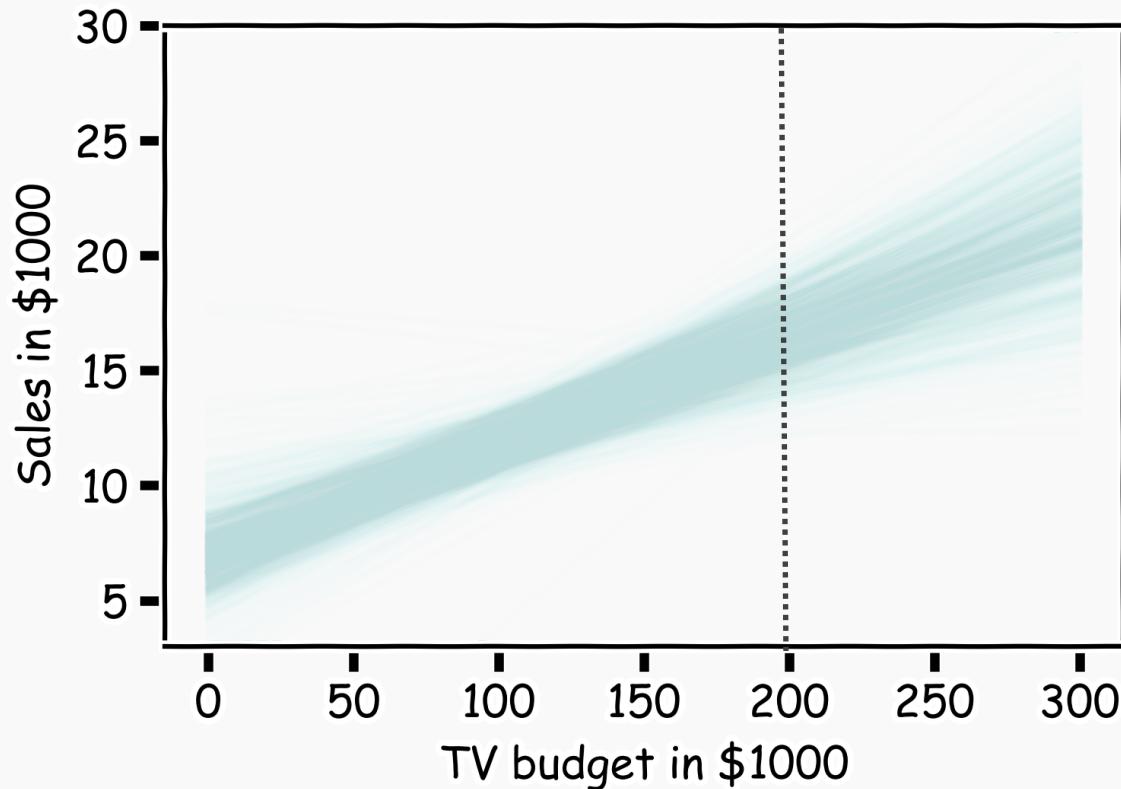
Below we show all regression lines for a thousand of such bootstrapped samples.

For a given x , we examine the distribution of \hat{f} , and determine the mean and standard deviation.



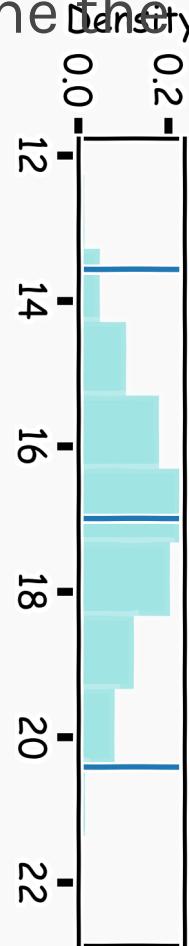
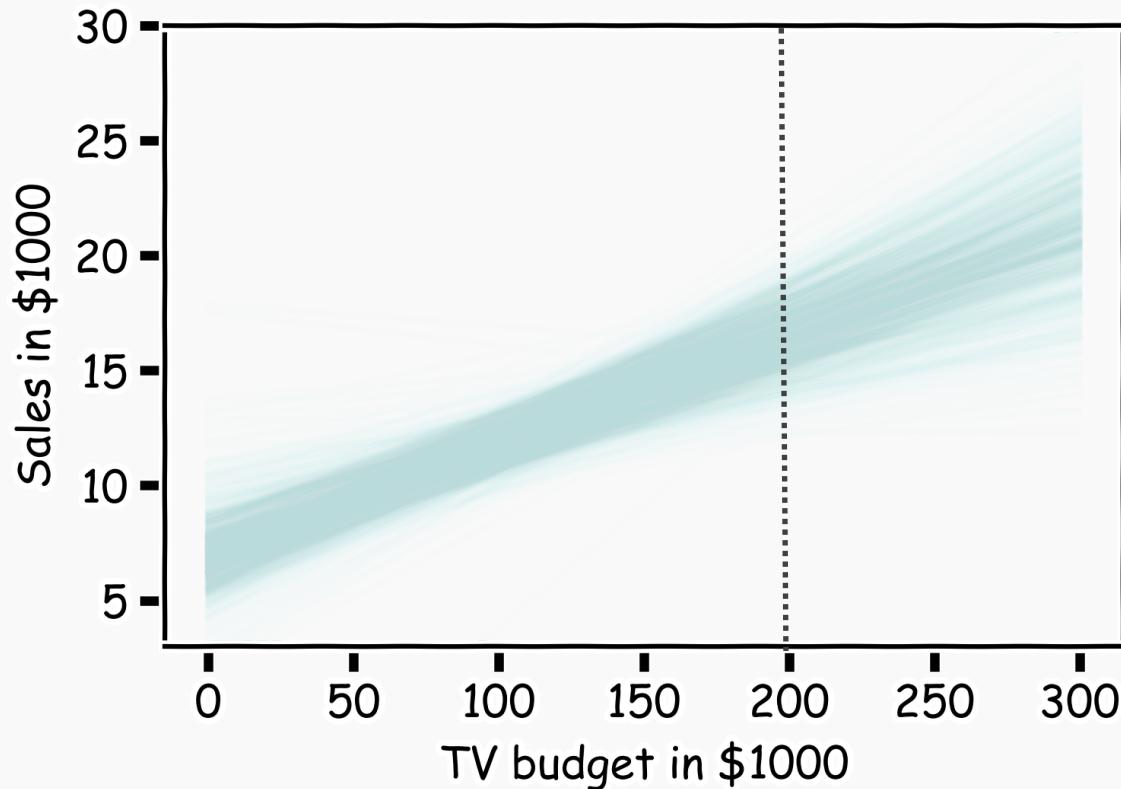
How well do we know \hat{f} ?

Below we show all regression lines for a thousand of such sub-samples. For a given x , we examine the distribution of \hat{f} , and determine the mean and standard deviation.



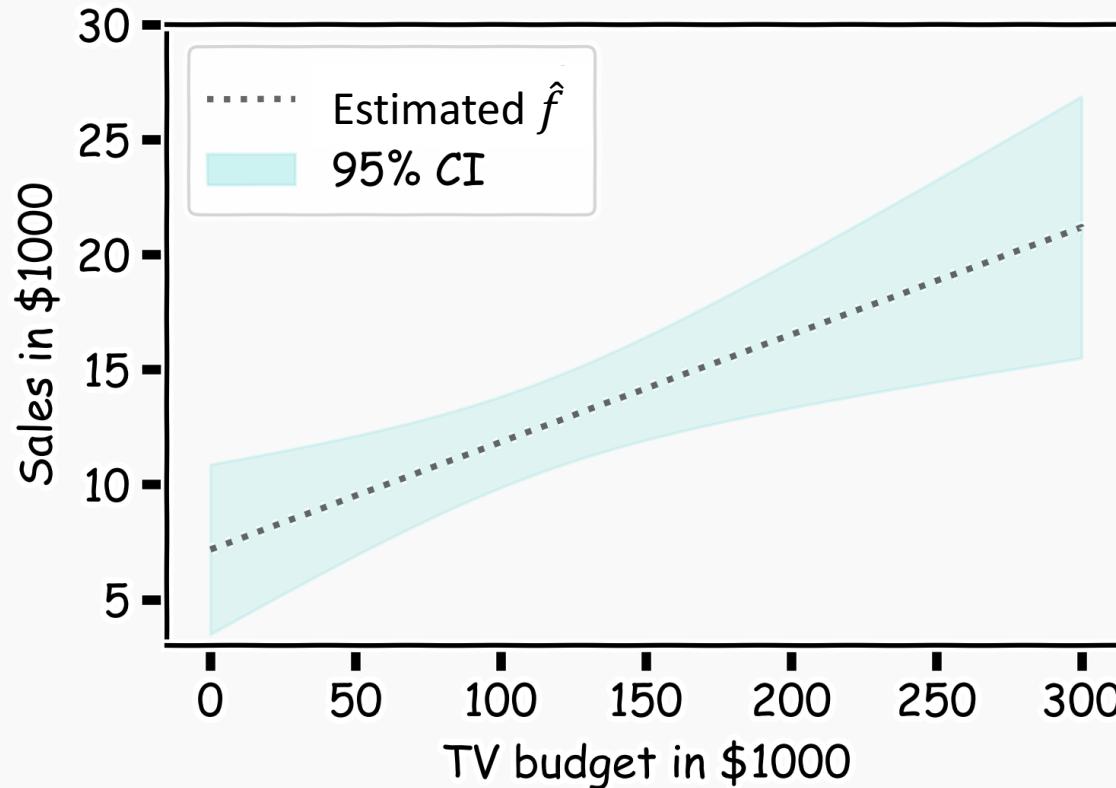
How well do we know \hat{f} ?

Below we show all regression lines for a thousand of such sub-samples. For a given x , we examine the distribution of \hat{f} , and determine the mean and standard deviation.

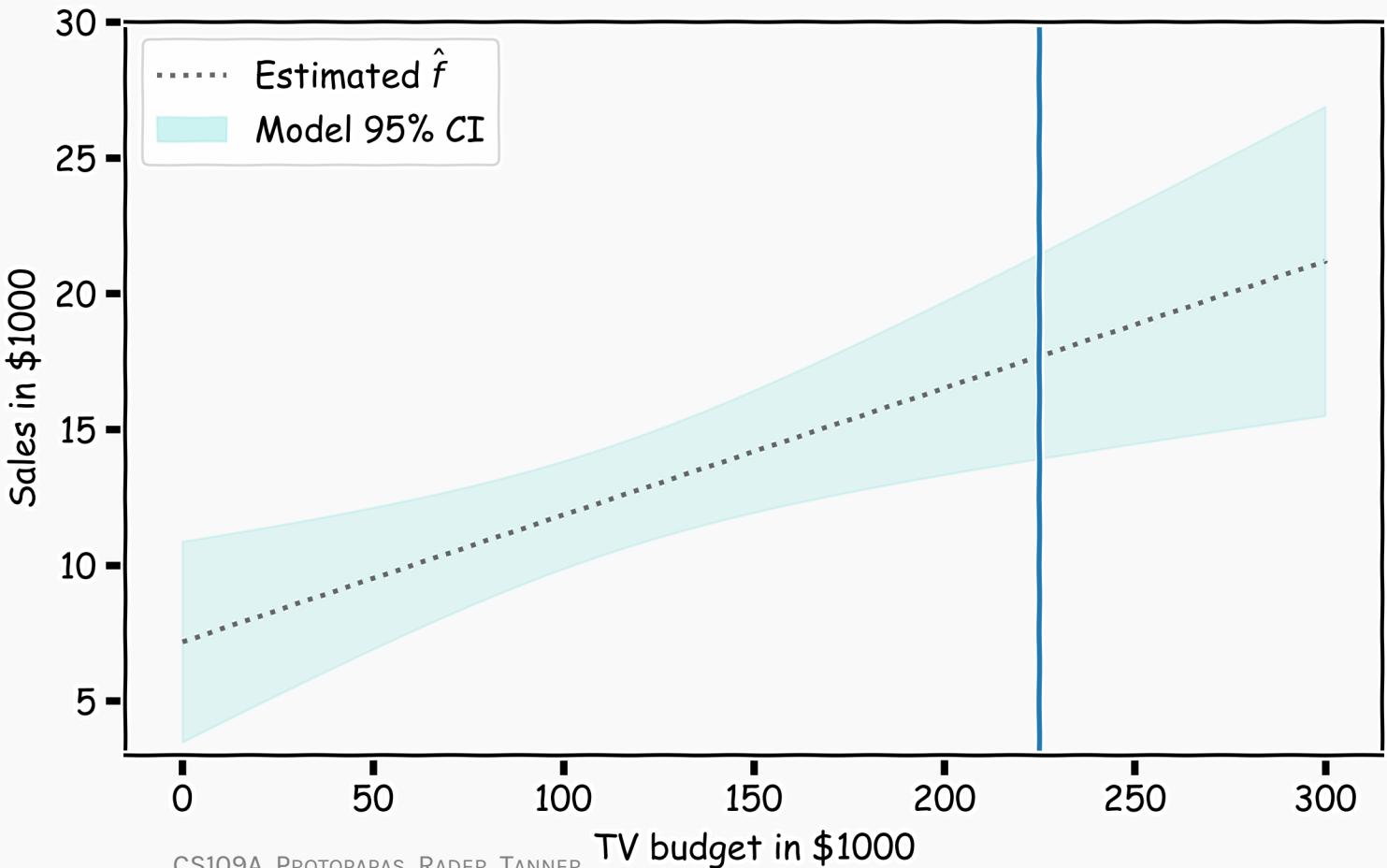


How well do we know \hat{f} ?

For every x , we calculate the mean of the models, \hat{f} (shown with dotted line) and the 95% CI of those models (shaded area).

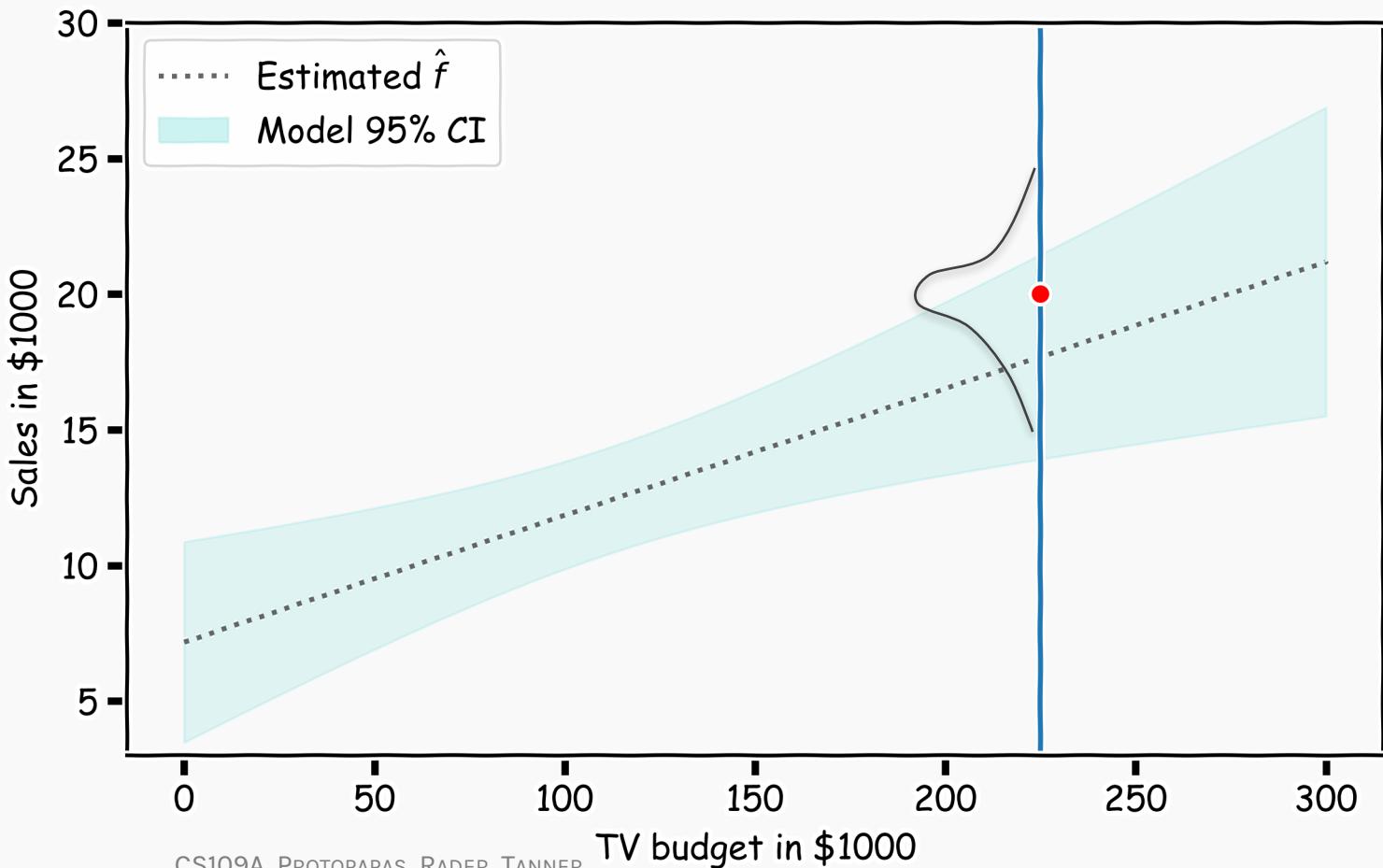


Confidence in predicting \hat{y}



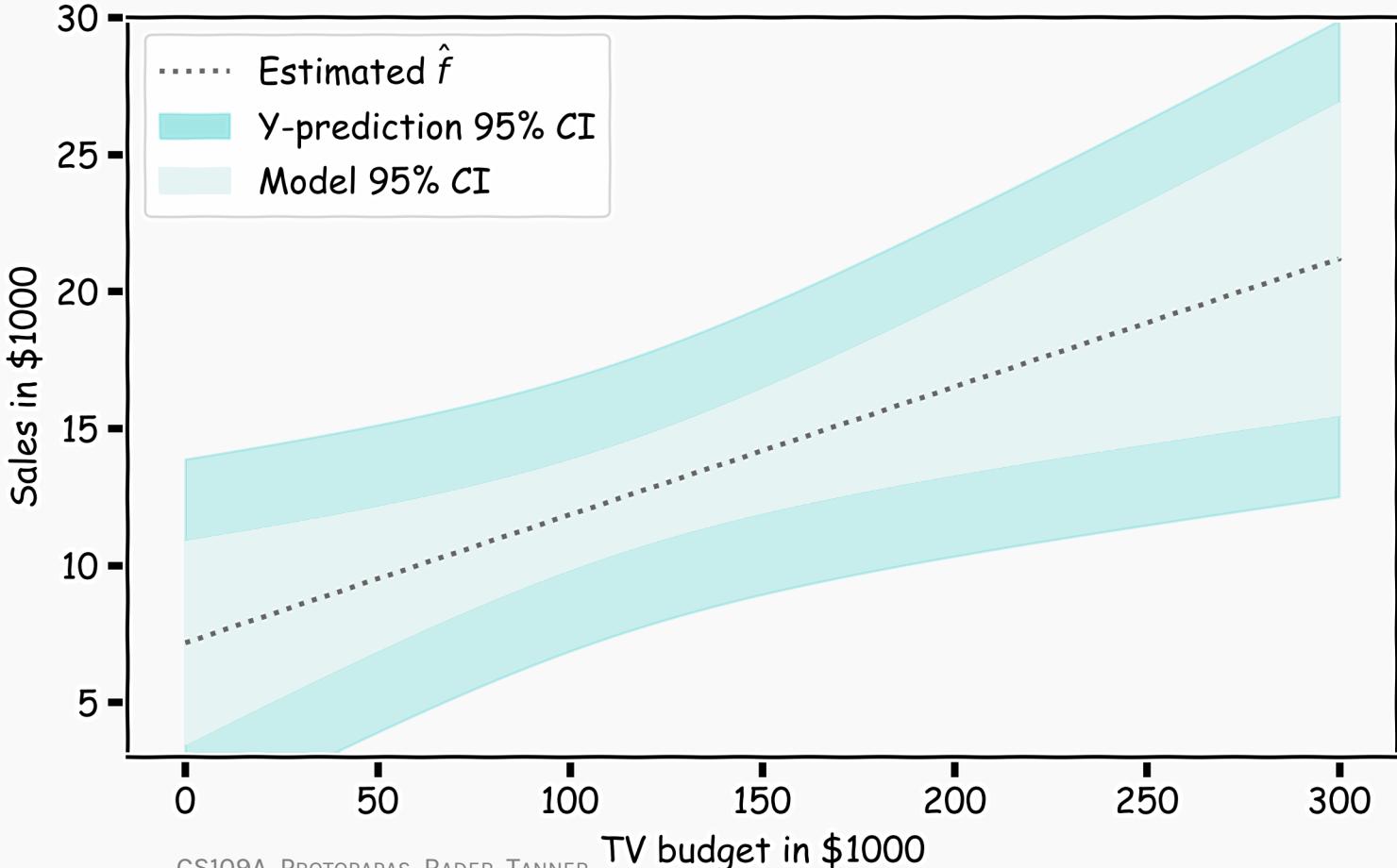
Confidence in predicting \hat{y}

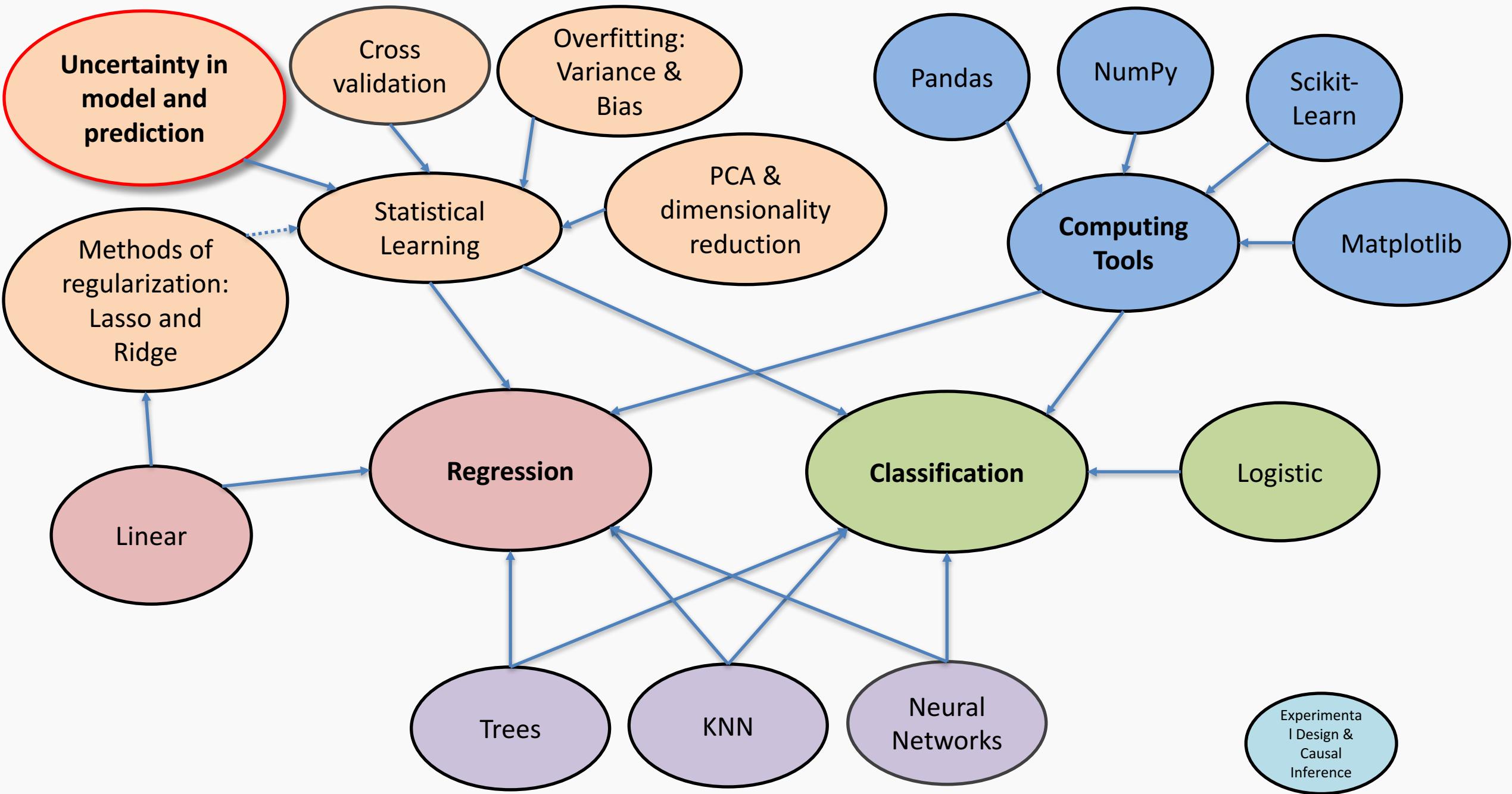
- for a given x , we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f, \sigma_\epsilon)$

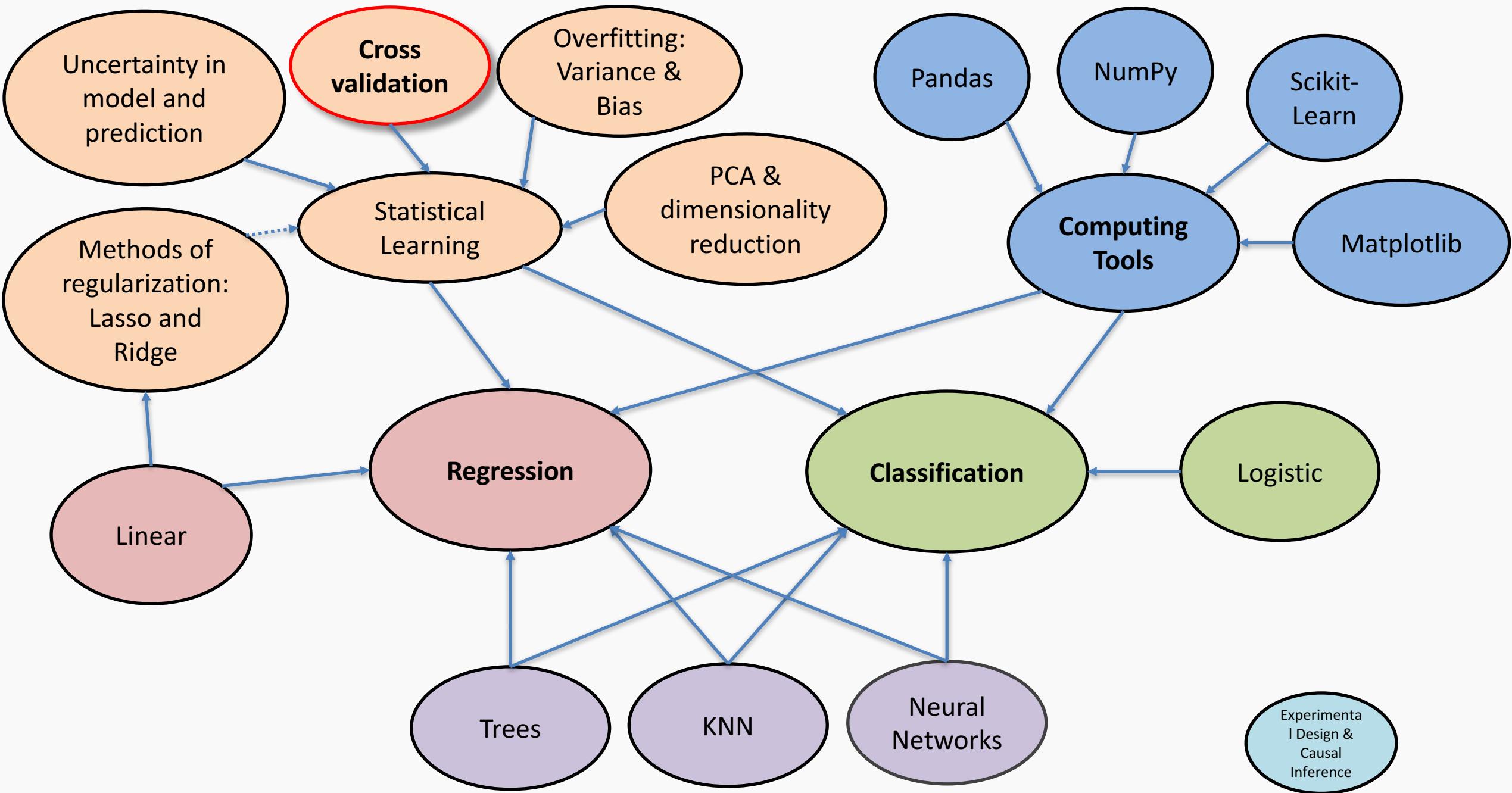


Confidence in predicting \hat{y}

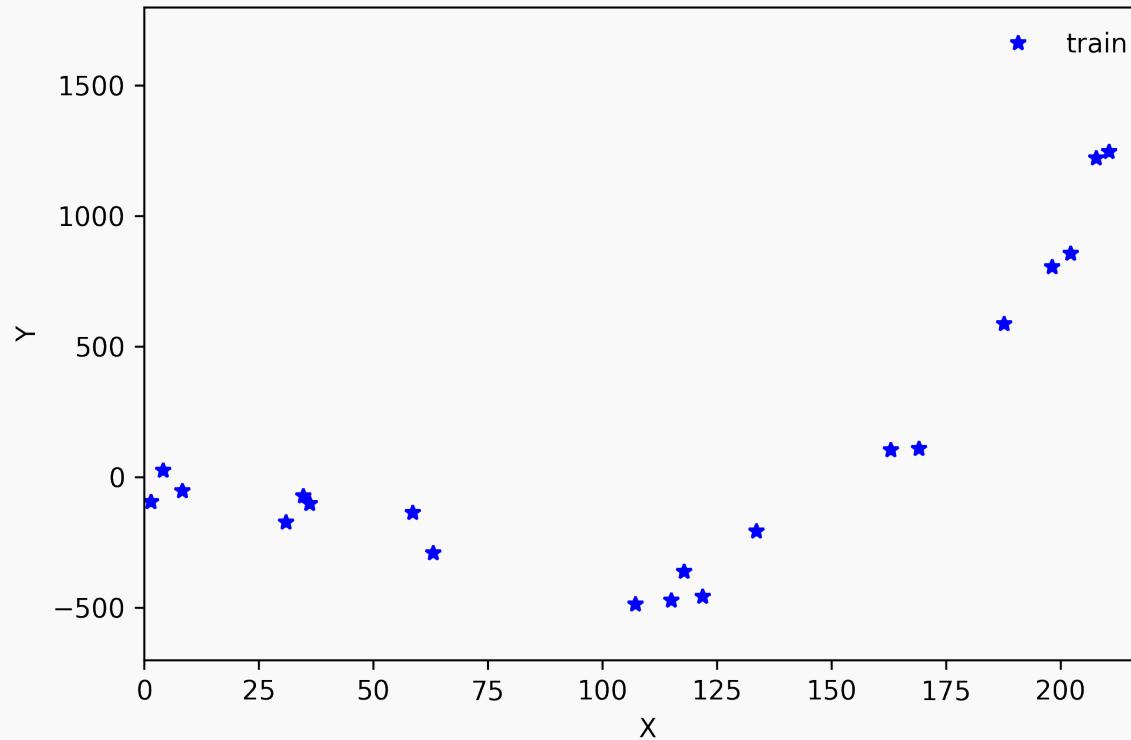
- for a given x , we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f, \sigma_\epsilon)$
- The prediction confidence intervals are then



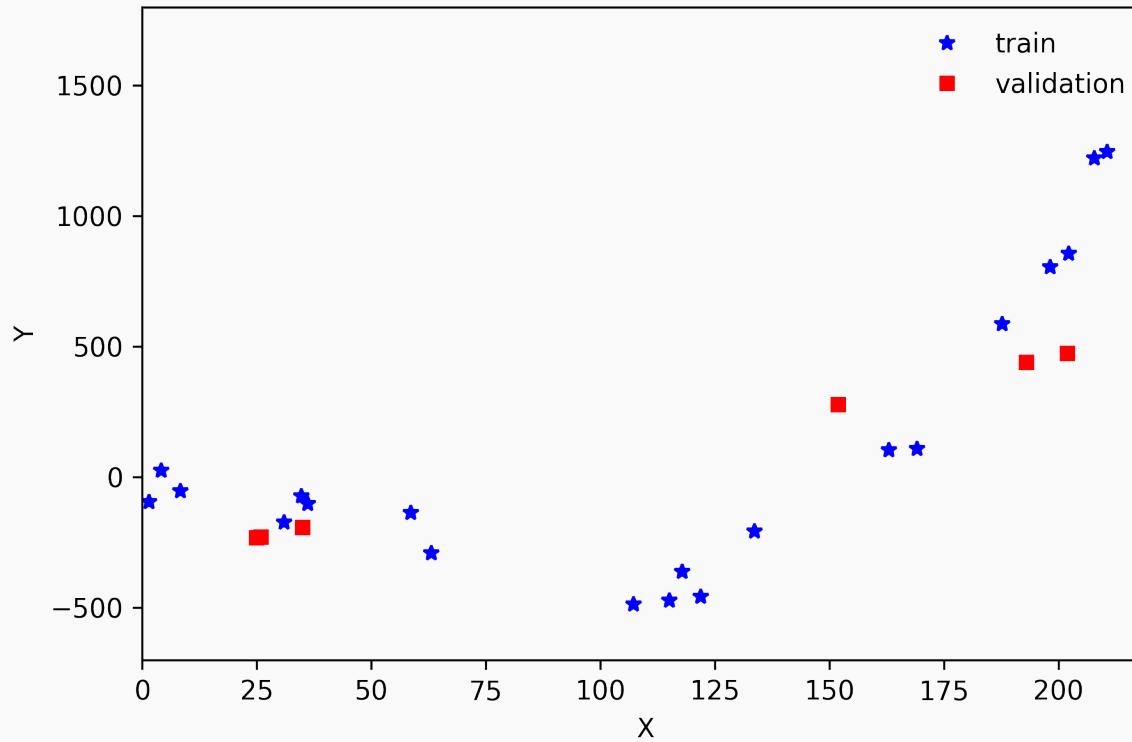




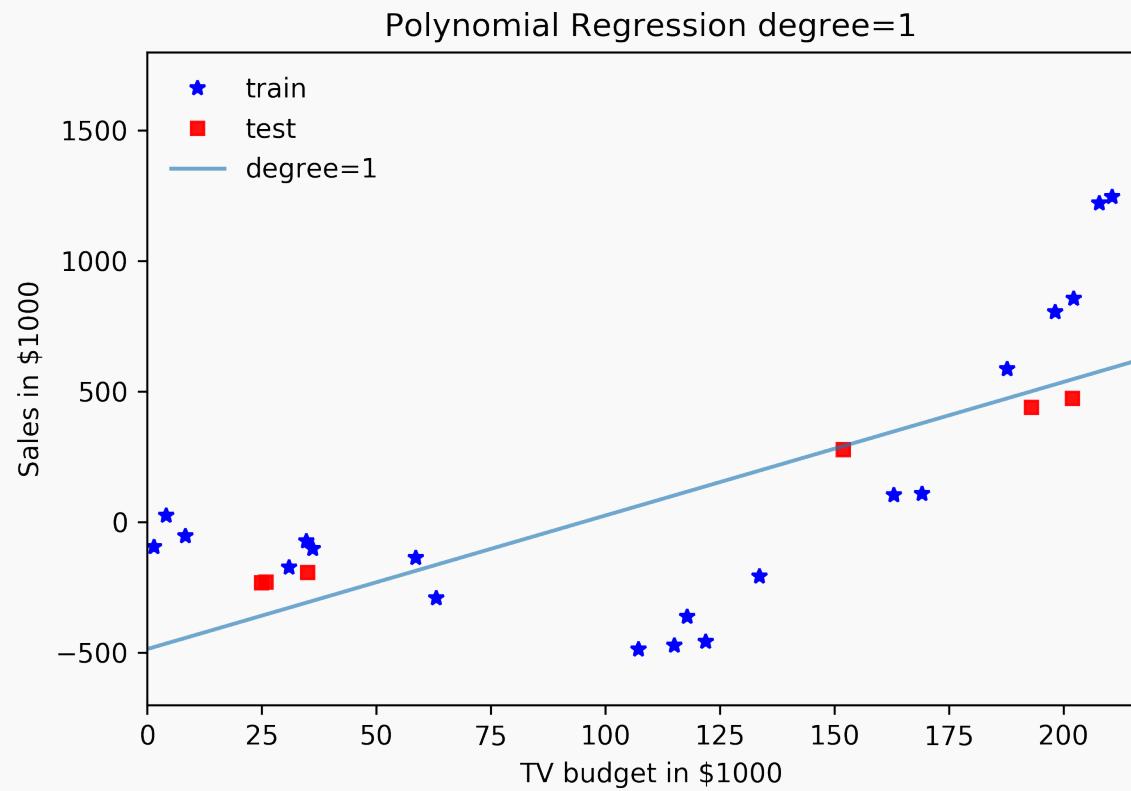
Cross Validation



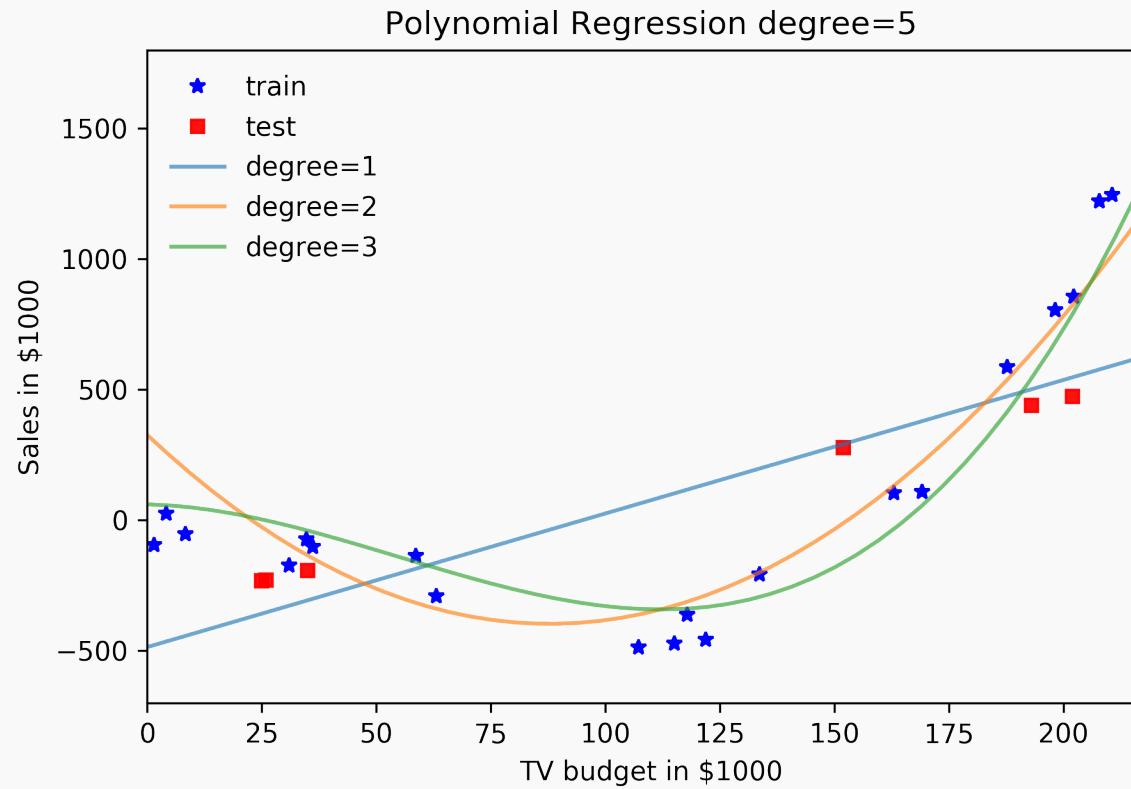
Cross Validation



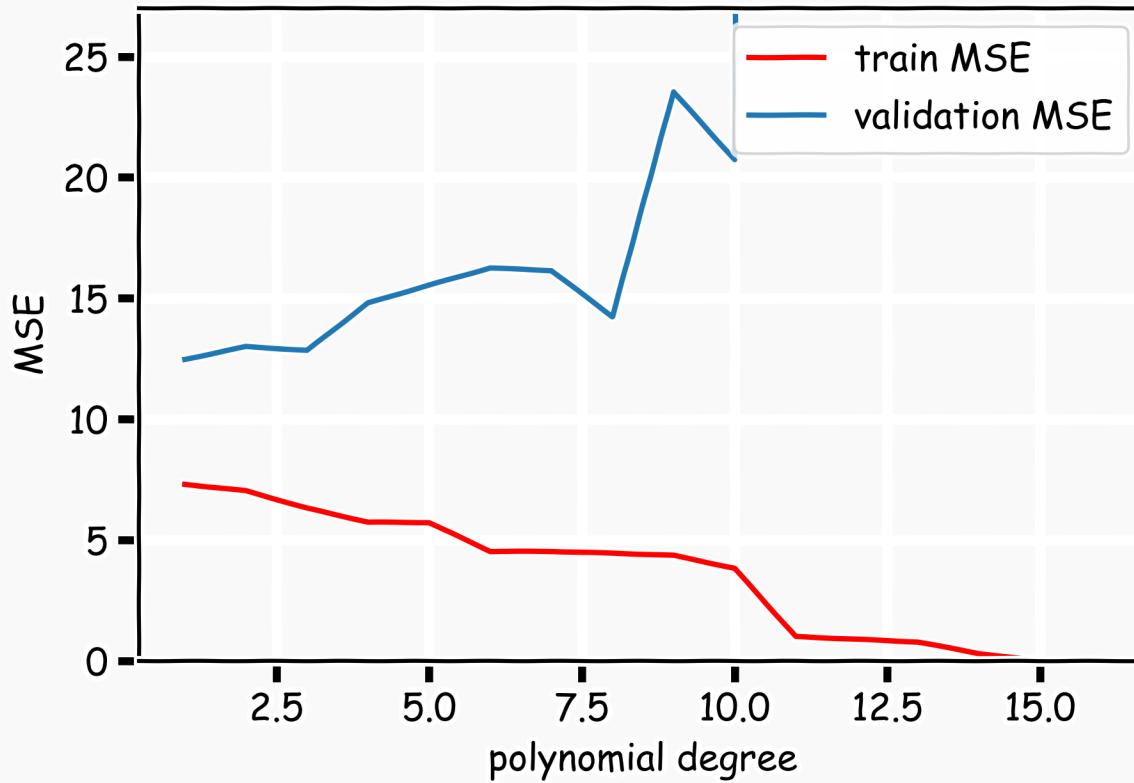
Cross Validation



Cross Validation

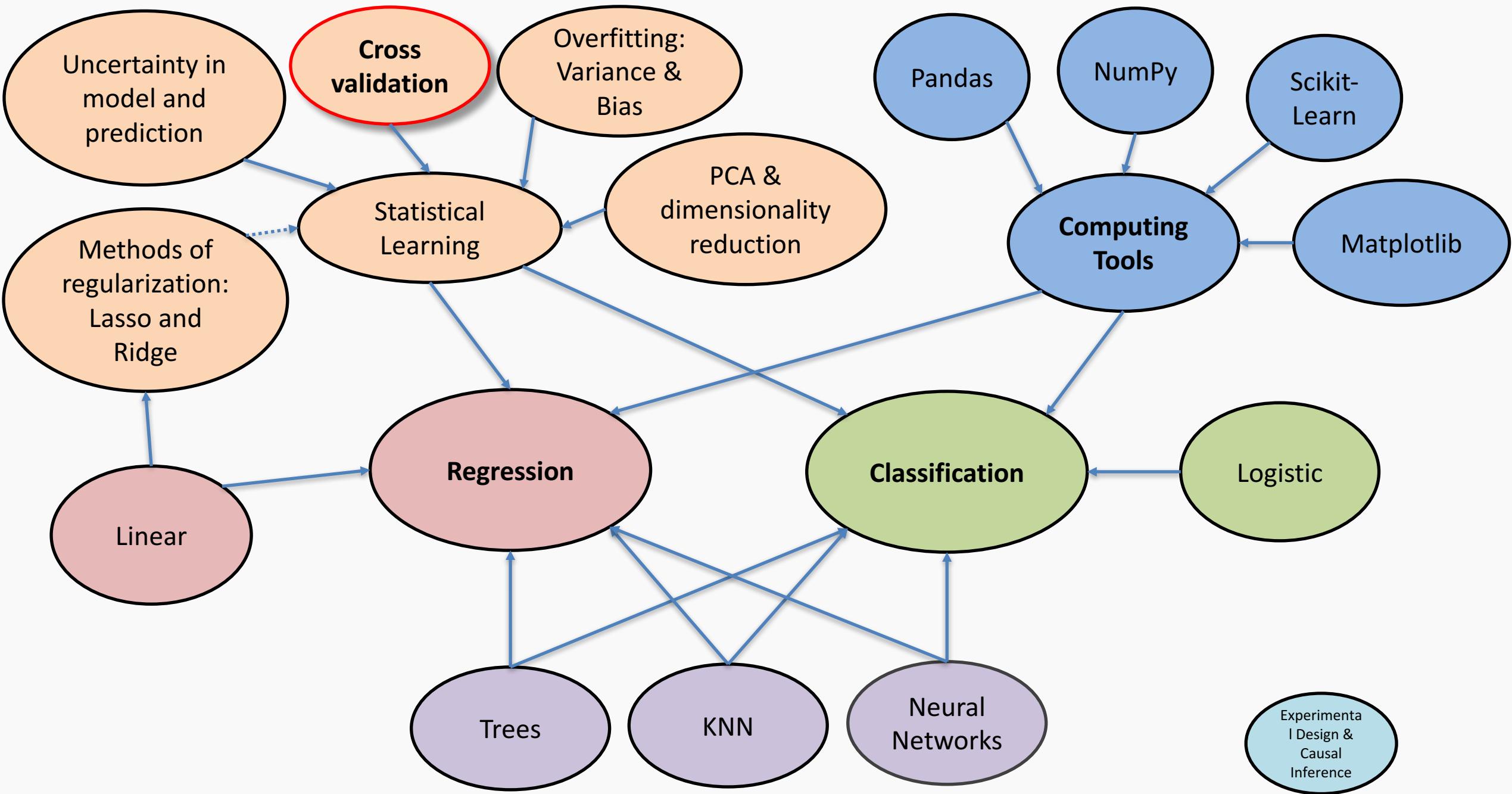


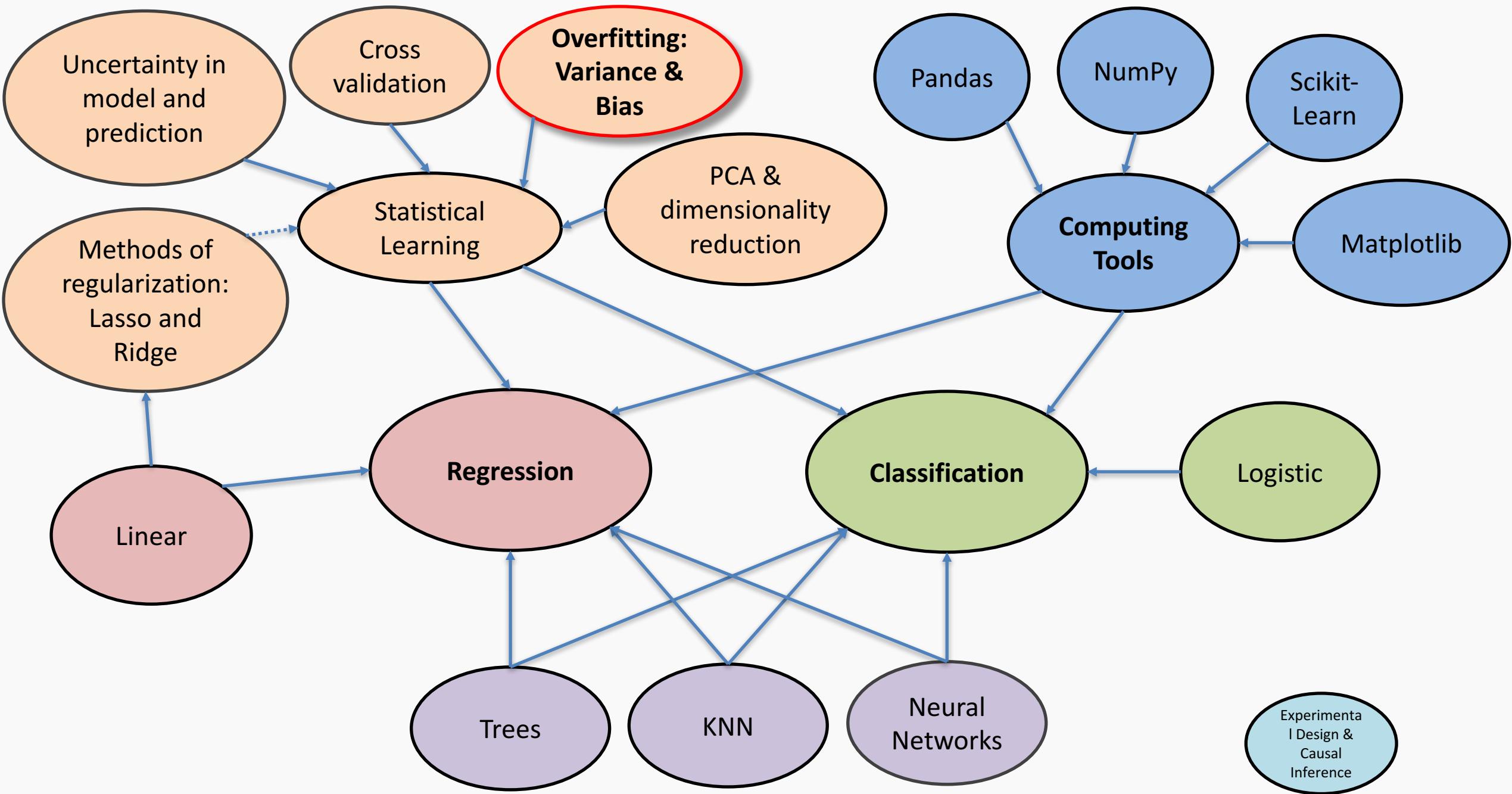
Validation

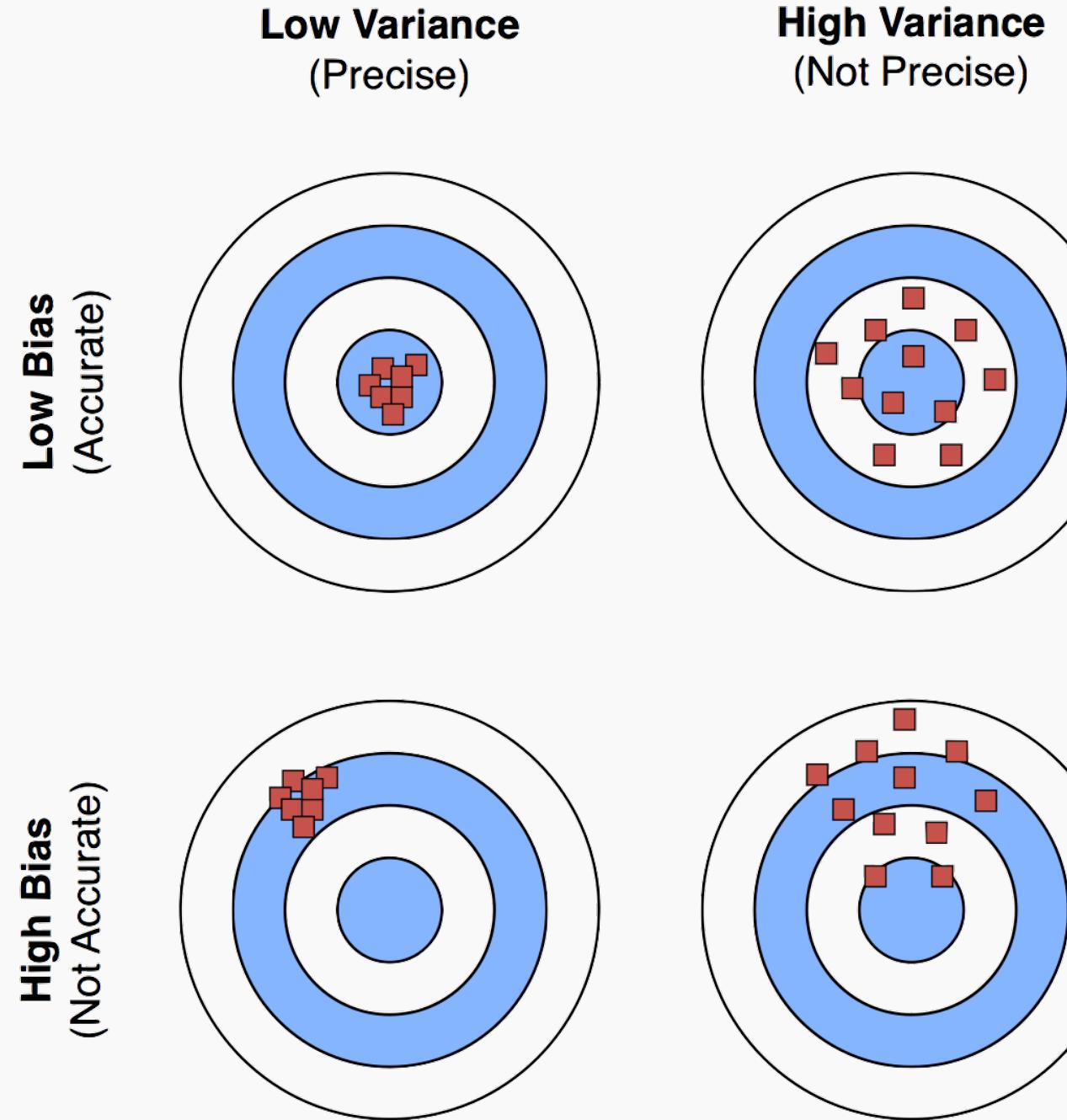


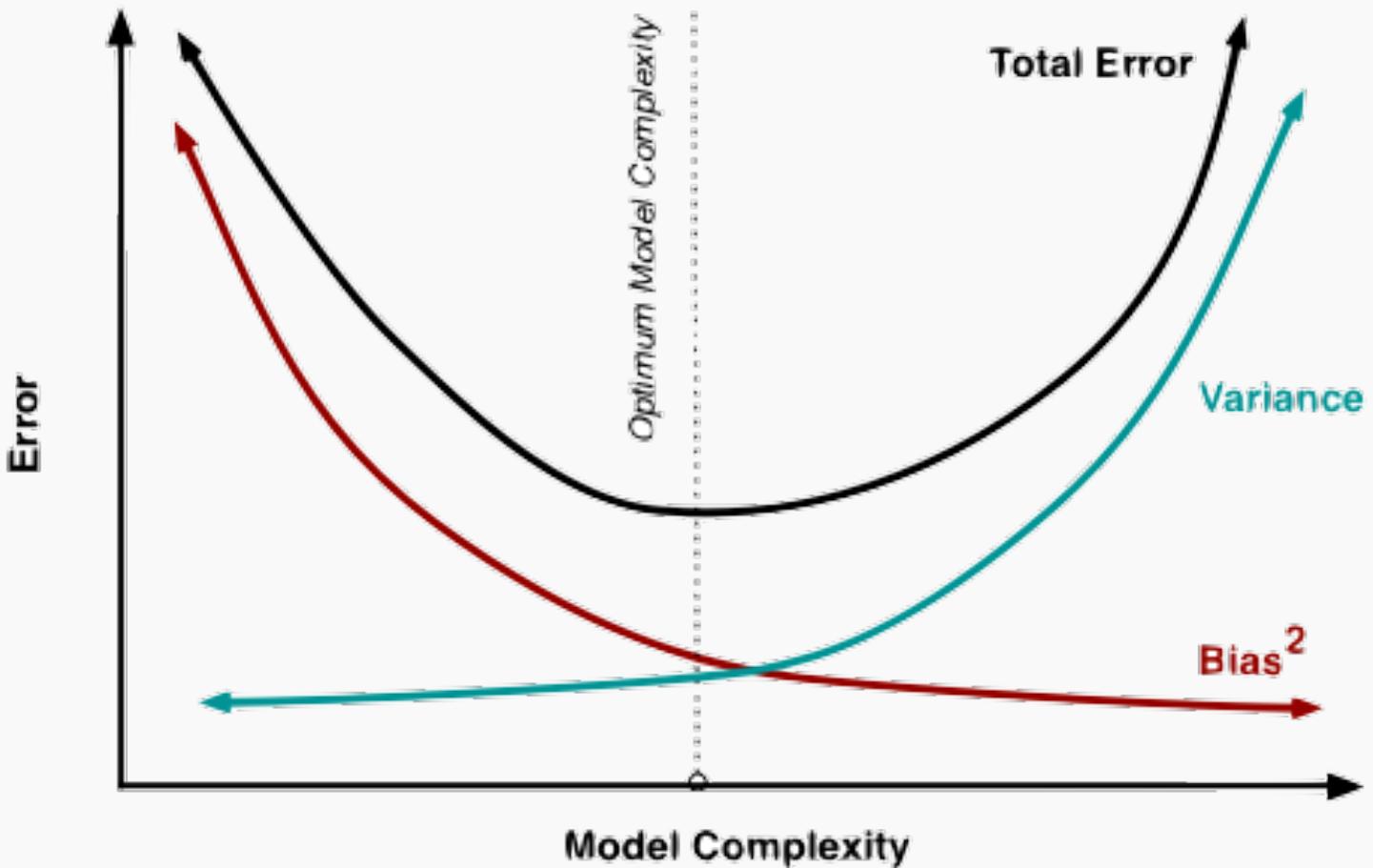
Cross Validation

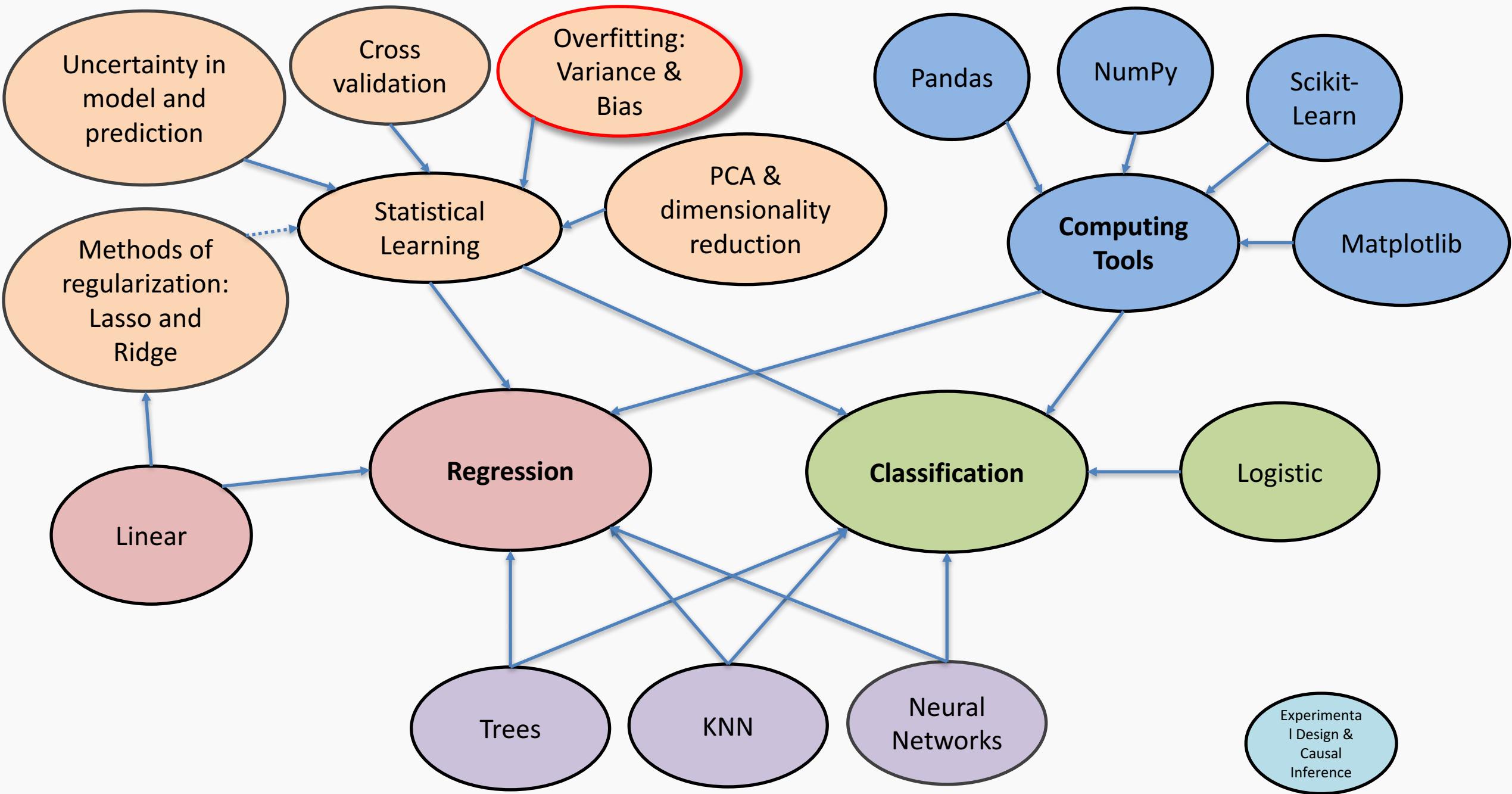


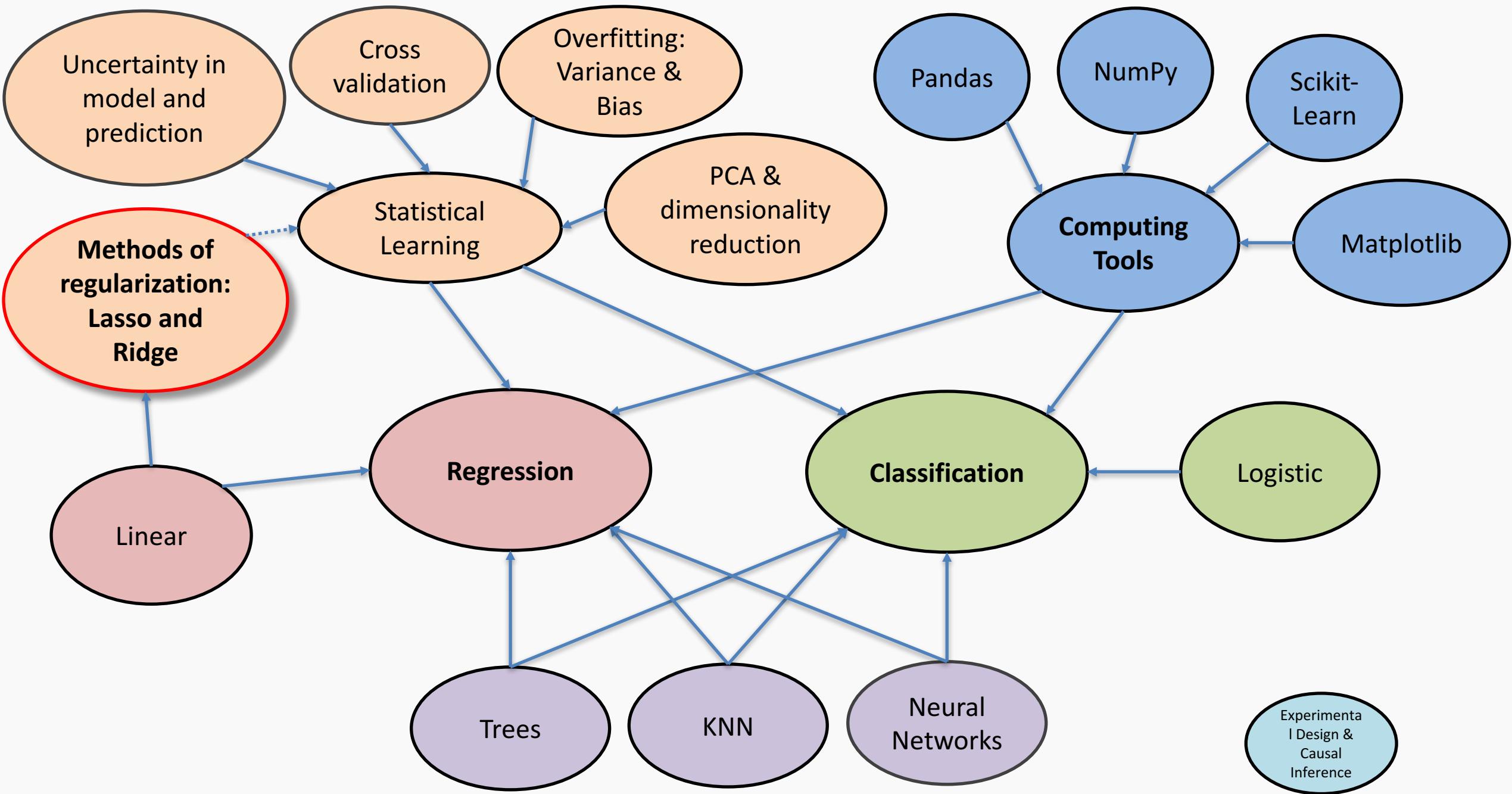




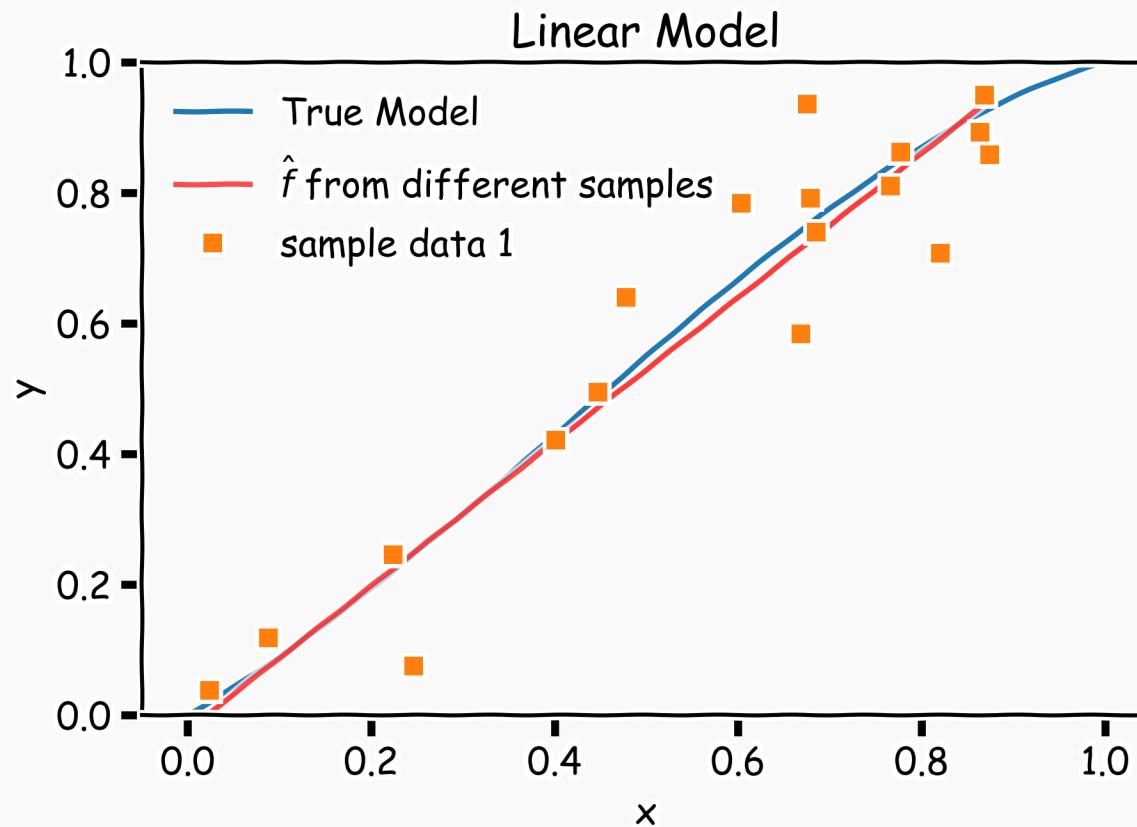




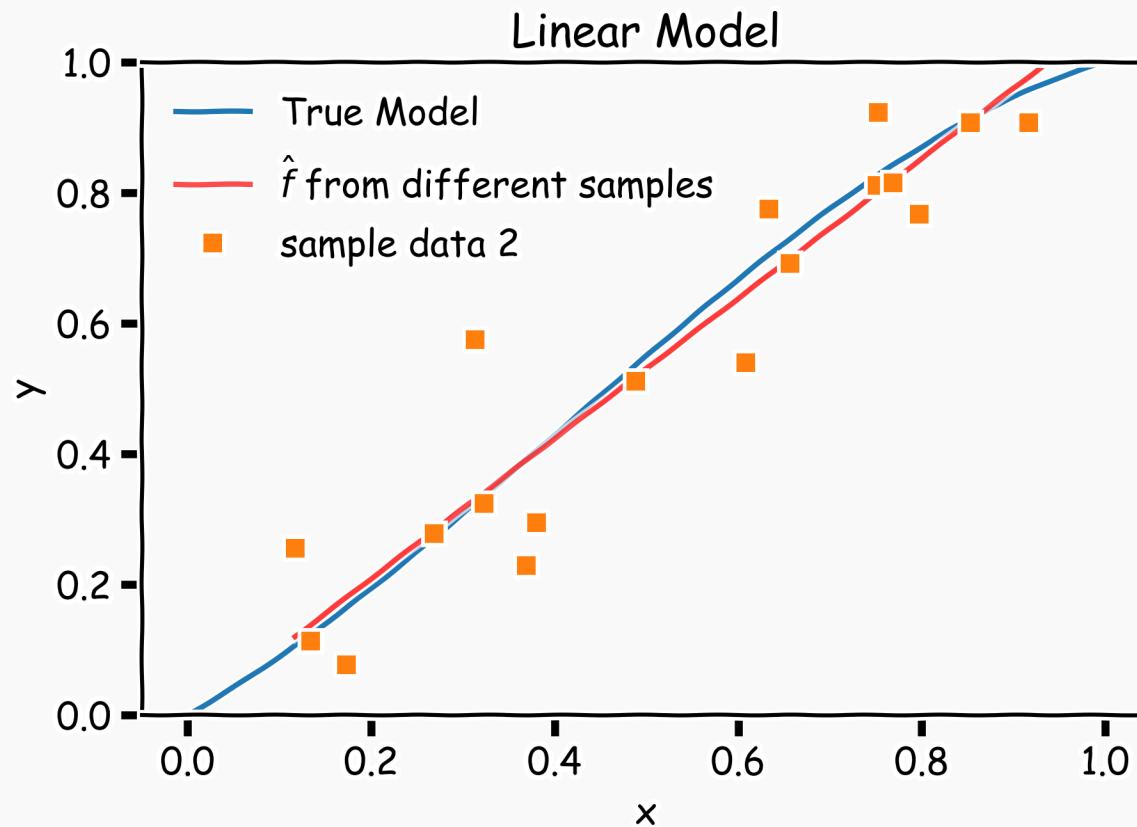




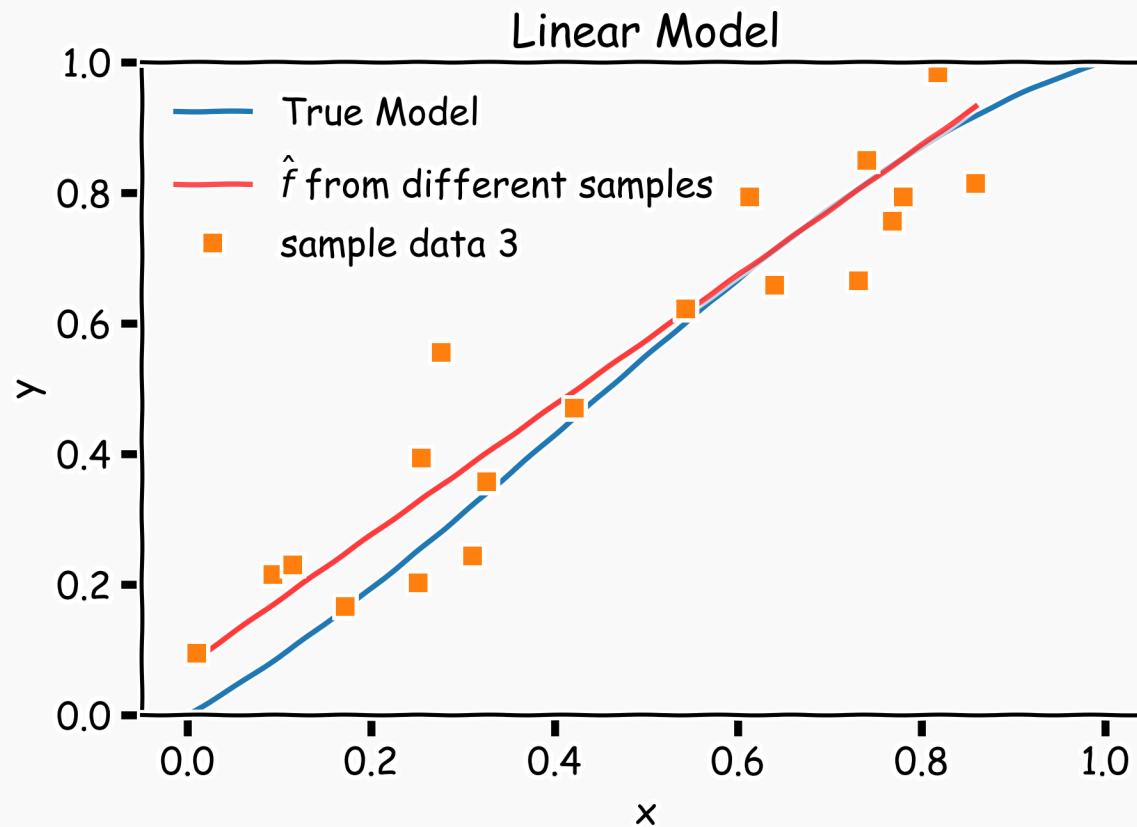
Bias vs Variance



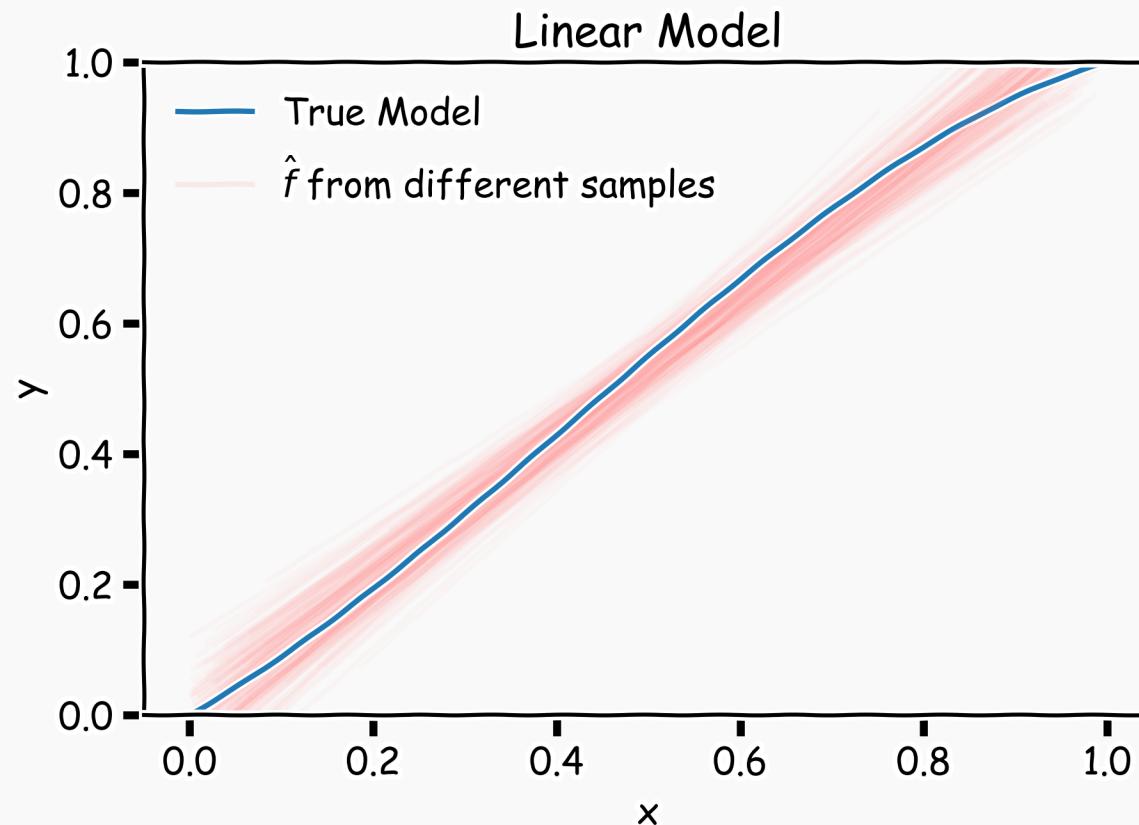
Bias vs Variance



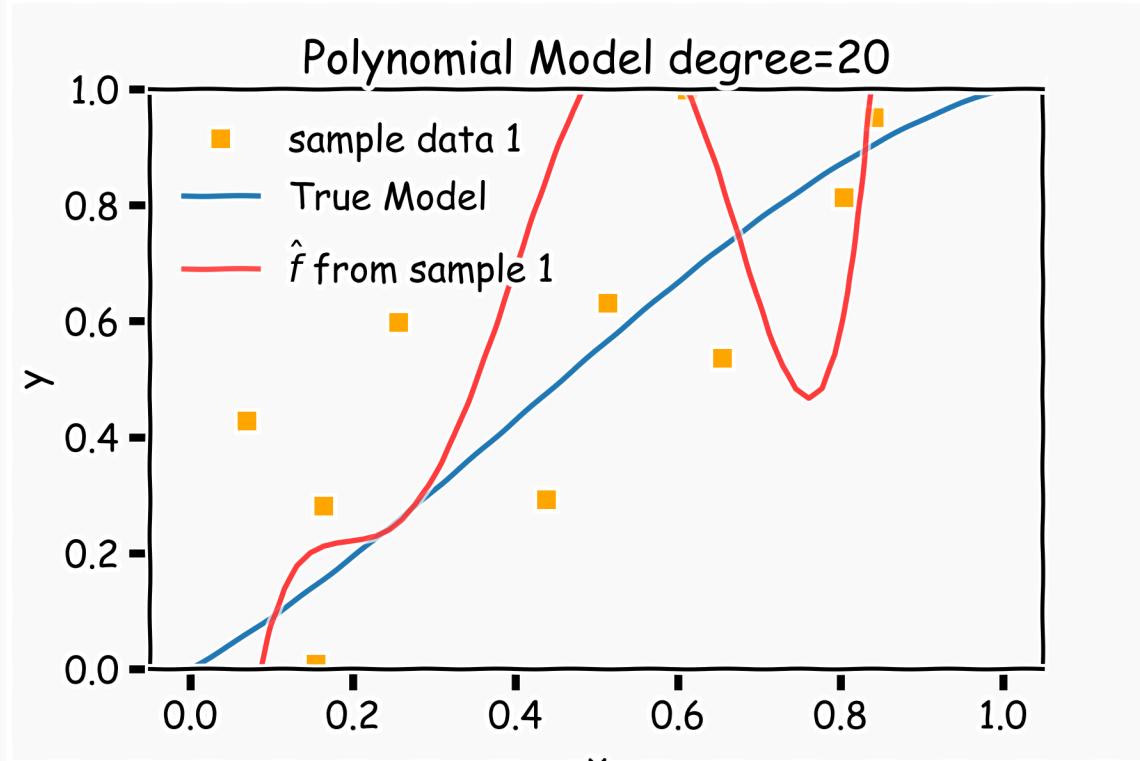
Bias vs Variance



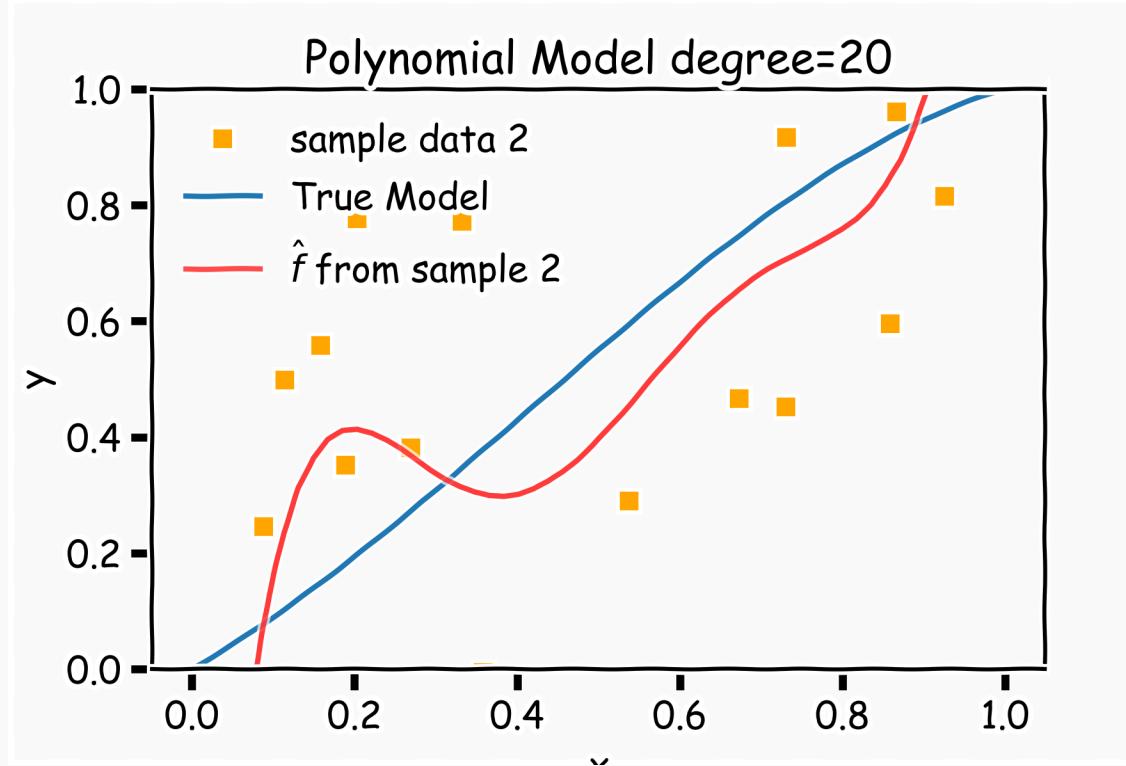
Linear models: 20 data points per line 2000 simulations



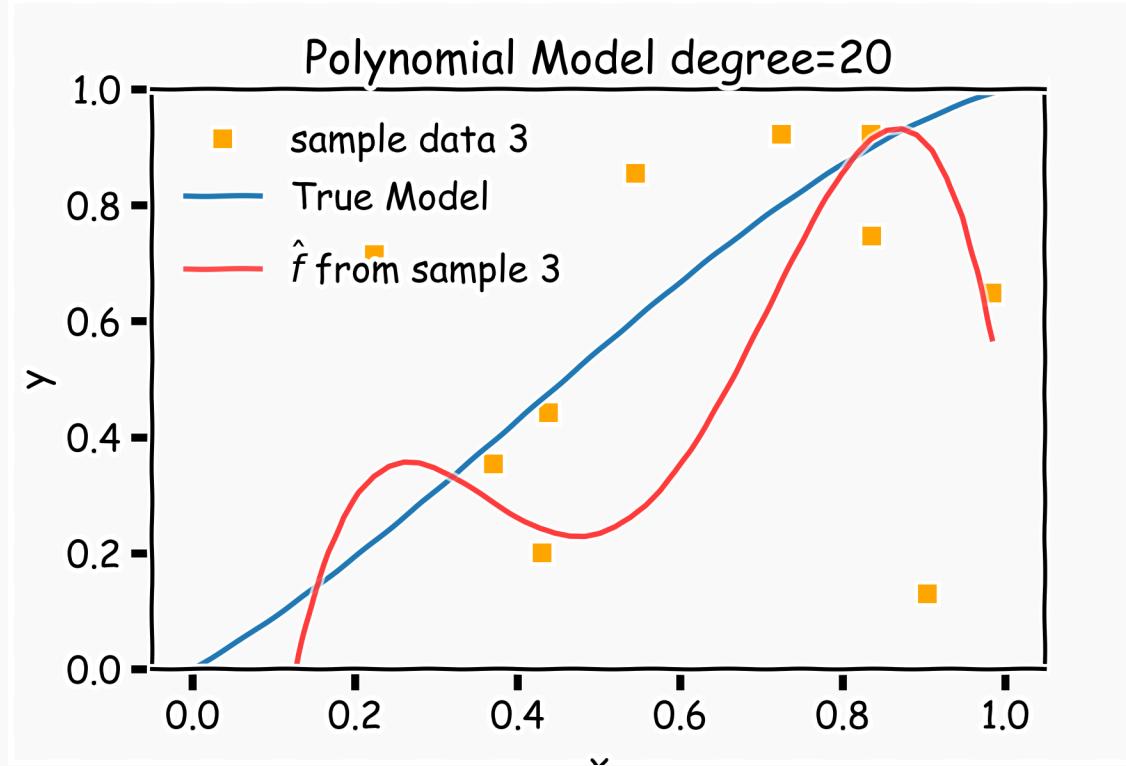
Bias vs Variance



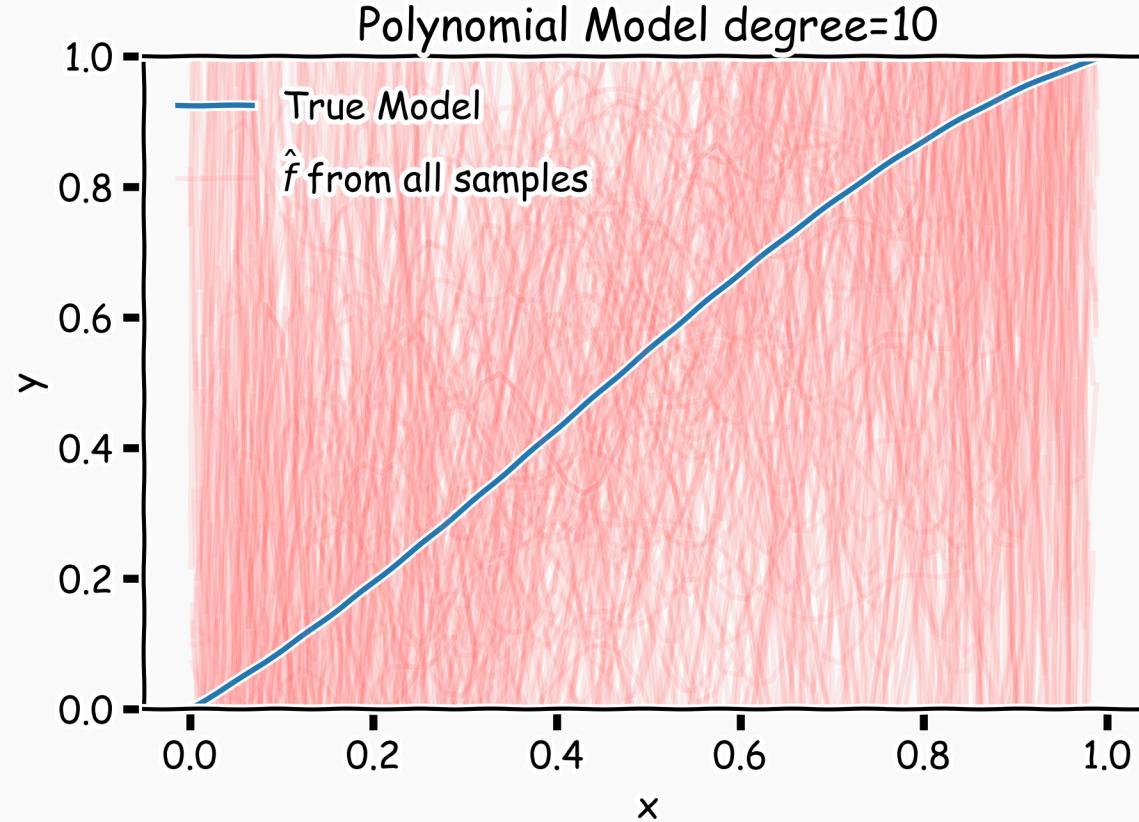
Bias vs Variance



Bias vs Variance



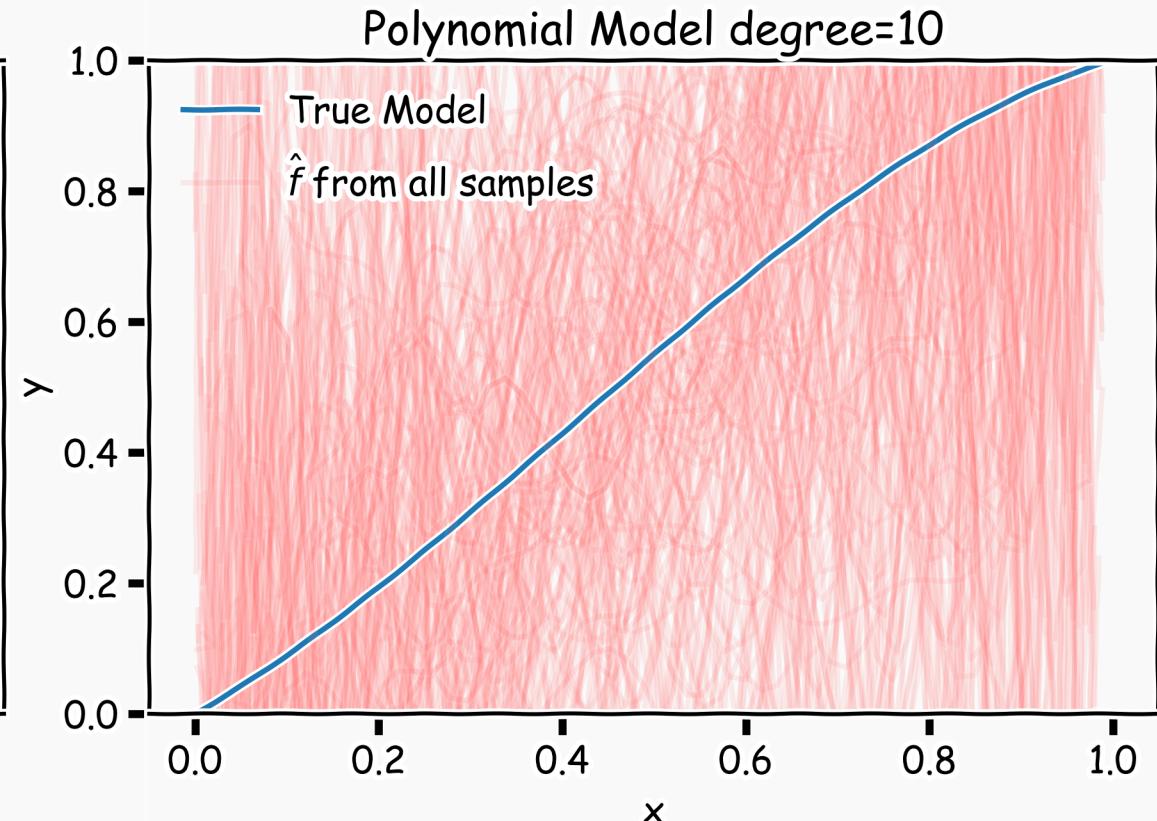
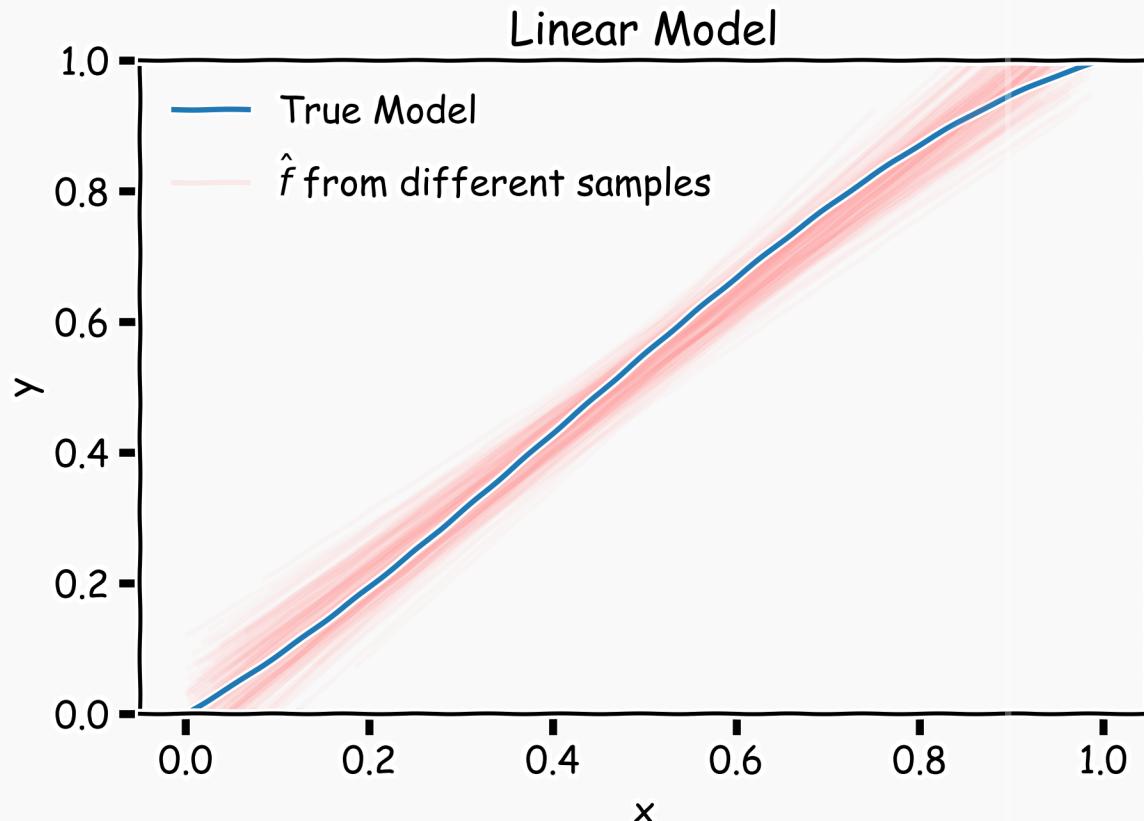
Poly 10 degree models : 20 data points per line 2000 simulations



Bias vs Variance

Left: 2000 best fit straight lines, each fitted on a different 20 point training set.

Right: Best-fit models using degree 10 polynomial



Regularization: An Overview

The idea of regularization revolves around modifying the loss function L ; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

where λ is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

LASSO Regression

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n} \sum_{i=1}^n |y_i - \beta^\top \mathbf{x}_i|^2 + \lambda \sum_{j=1}^J |\beta_j|.$$

Note that $\sum_{j=1}^J |\beta_j|$ is the l_1 norm of the vector β

$$\sum_{j=1}^J |\beta_j| = \|\beta\|_1$$



Ridge Regression

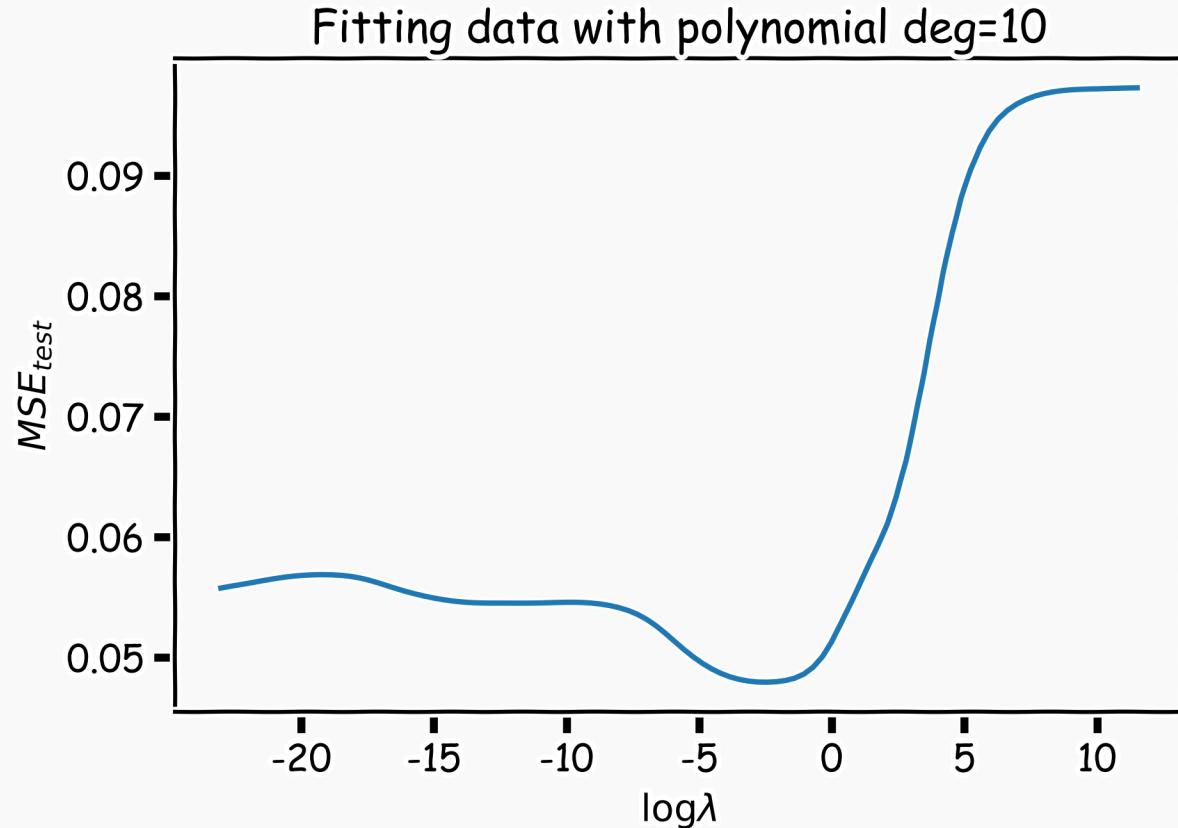
Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n} \sum_{i=1}^n |y_i - \boldsymbol{\beta}^\top \mathbf{x}_i|^2 + \lambda \sum_{j=1}^J \beta_j^2.$$

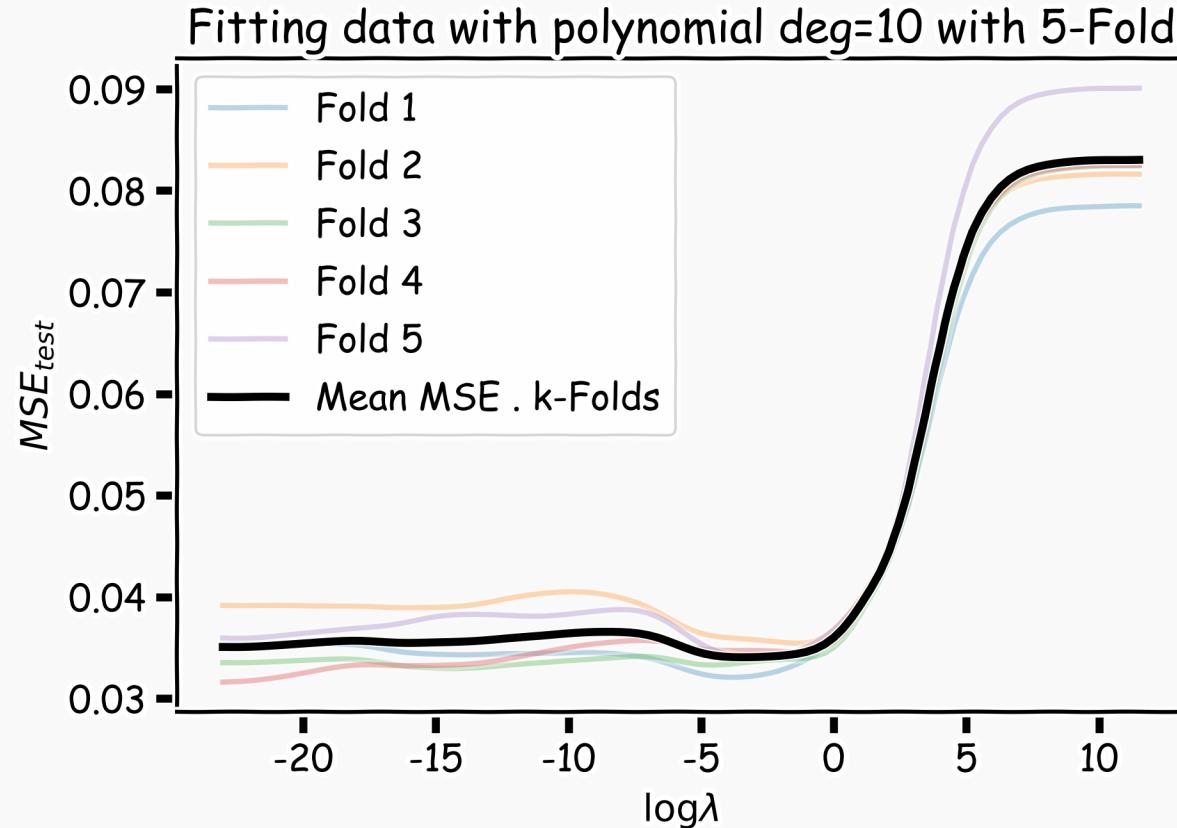
Note that $\sum_{j=1}^J \beta_j^2$ is the l_2 norm of the vector $\boldsymbol{\beta}$

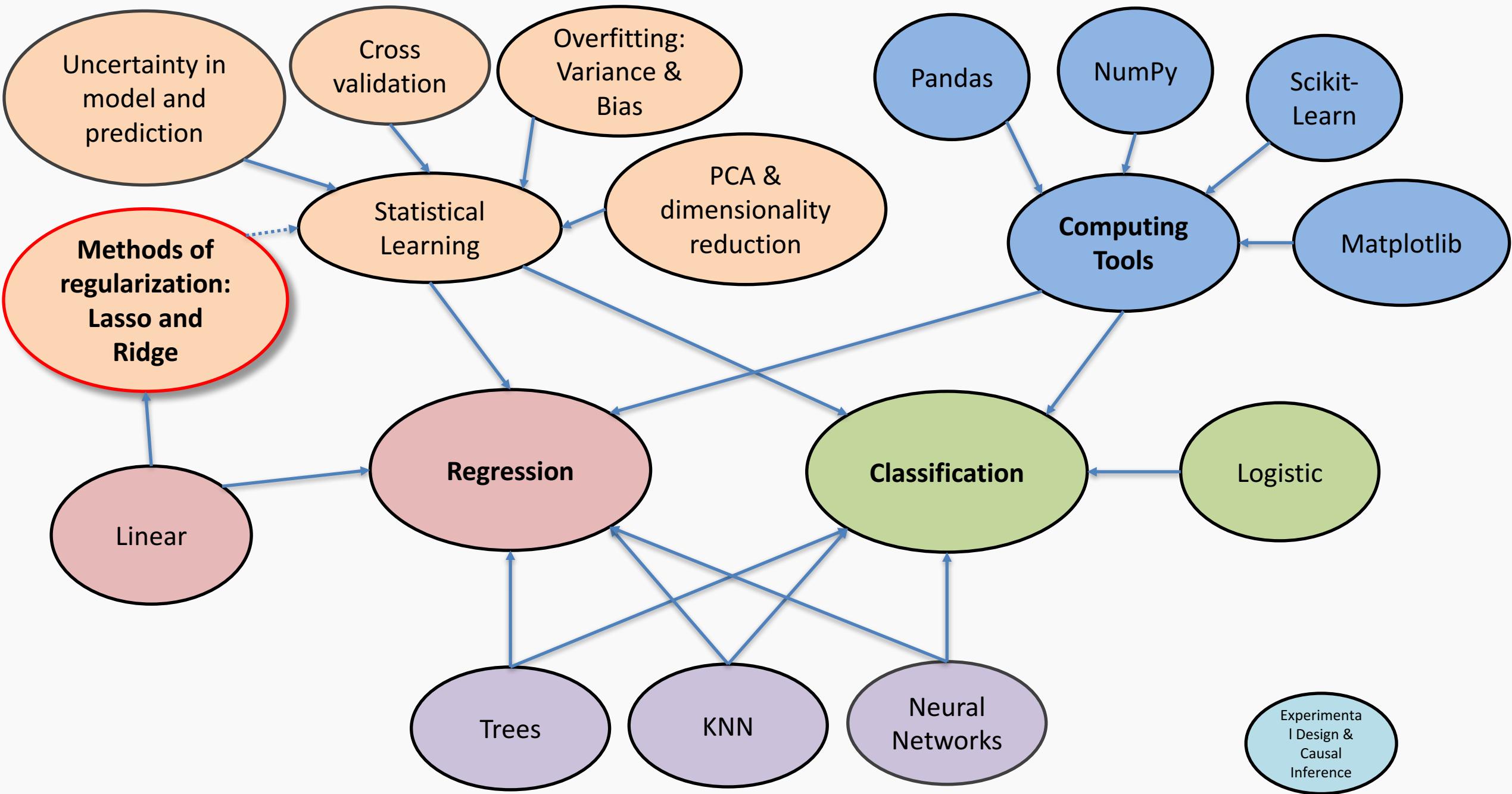
$$\sum_{j=1}^J \beta_j^2 = \|\boldsymbol{\beta}\|_2^2$$

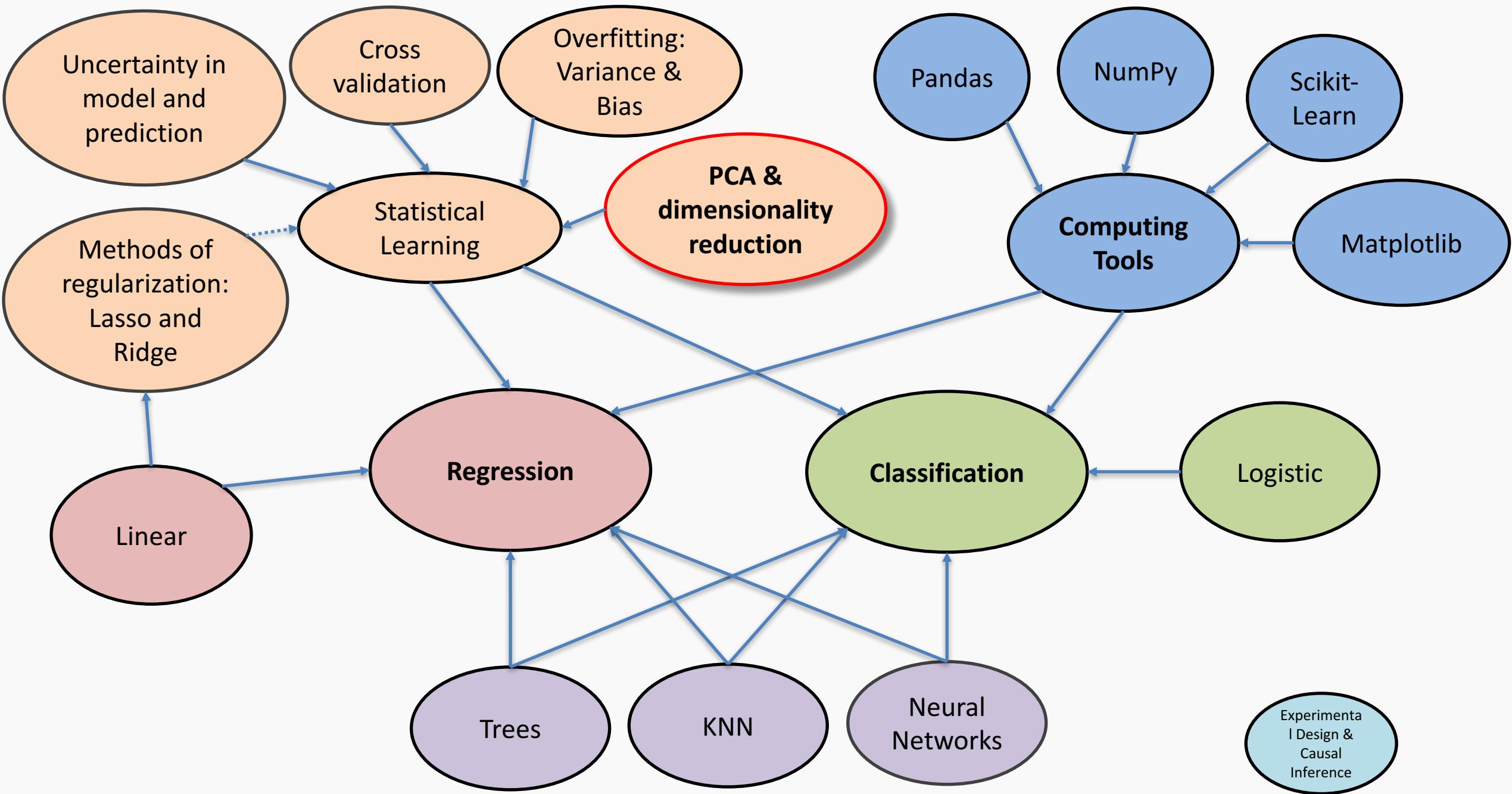
Ridge regularization with validation only: step by step



Ridge regularization with validation only: step by step



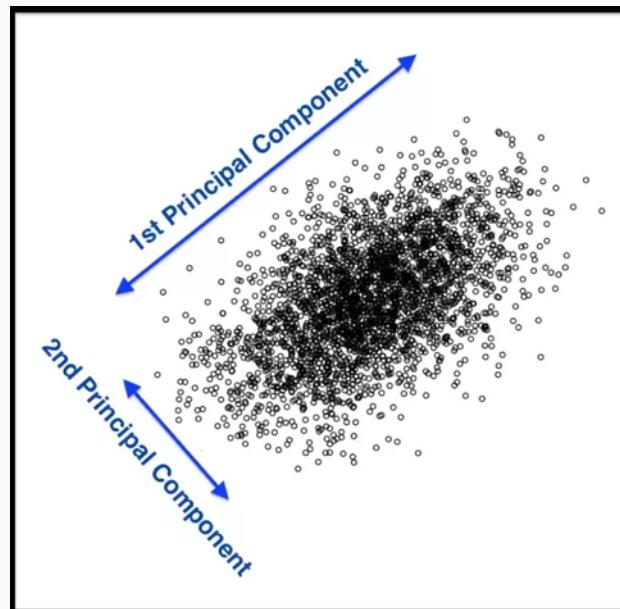




The Intuition Behind PCA

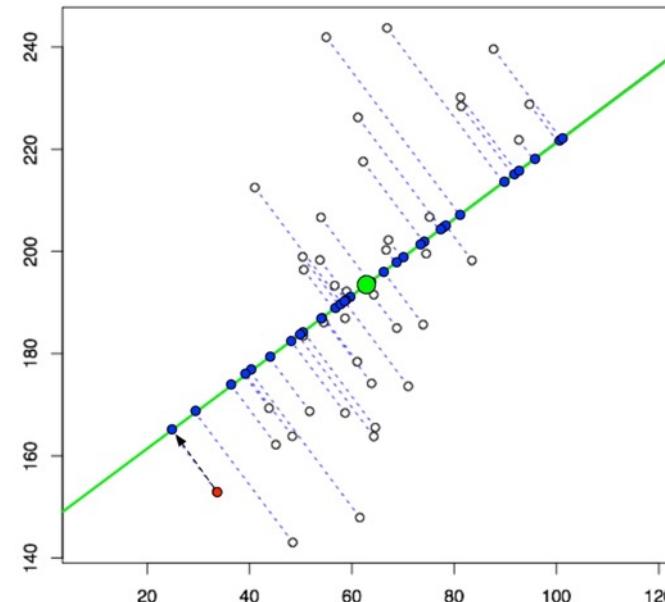
Top PCA components capture the most amount of variation (interesting features) of the data.

Each component is a linear combination of the original predictors - we visualize them as vectors in the feature space.



The Intuition Behind PCA (cont.)

Transforming our observed data means projecting our dataset onto the space defined by the top m PCA components, these components are our new predictors.



A Framework For Dimensionality Reduction

One way to reduce the dimensions of the feature space is to create a new, smaller set of predictors by taking linear combinations of the original predictors.

We choose Z_1, Z_2, \dots, Z_m , where $m \leq p$ and where each Z_i is a linear combination of the original p predictors

$$Z_i = \sum_{j=1}^p \phi_{ji} X_j$$

for fixed constants ϕ_{ji} . Then we can build a linear regression model using the new predictors

$$Y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_m Z_m + \varepsilon$$

Notice that this model has a smaller number ($m+1 < p+1$) of parameters.

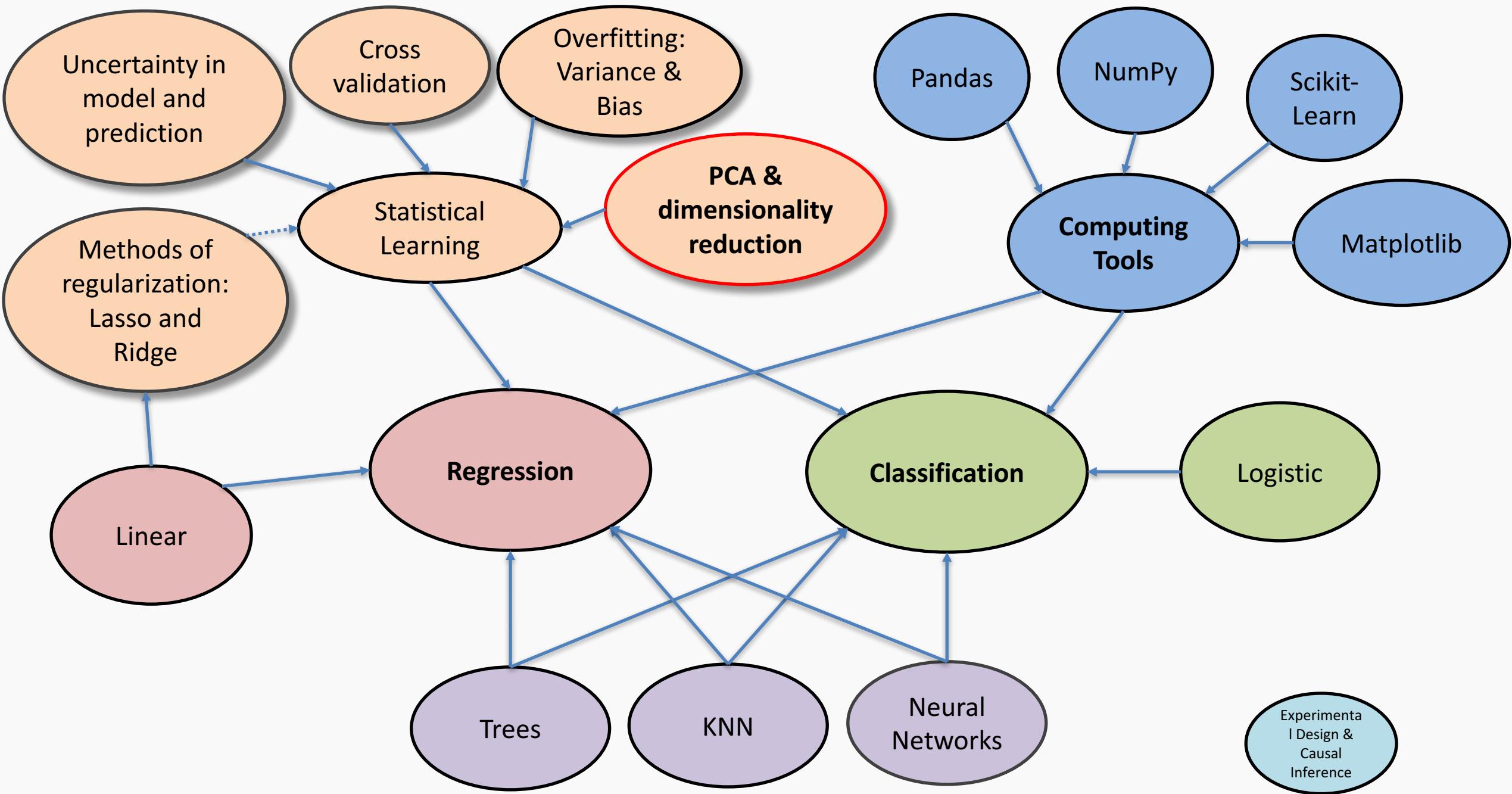


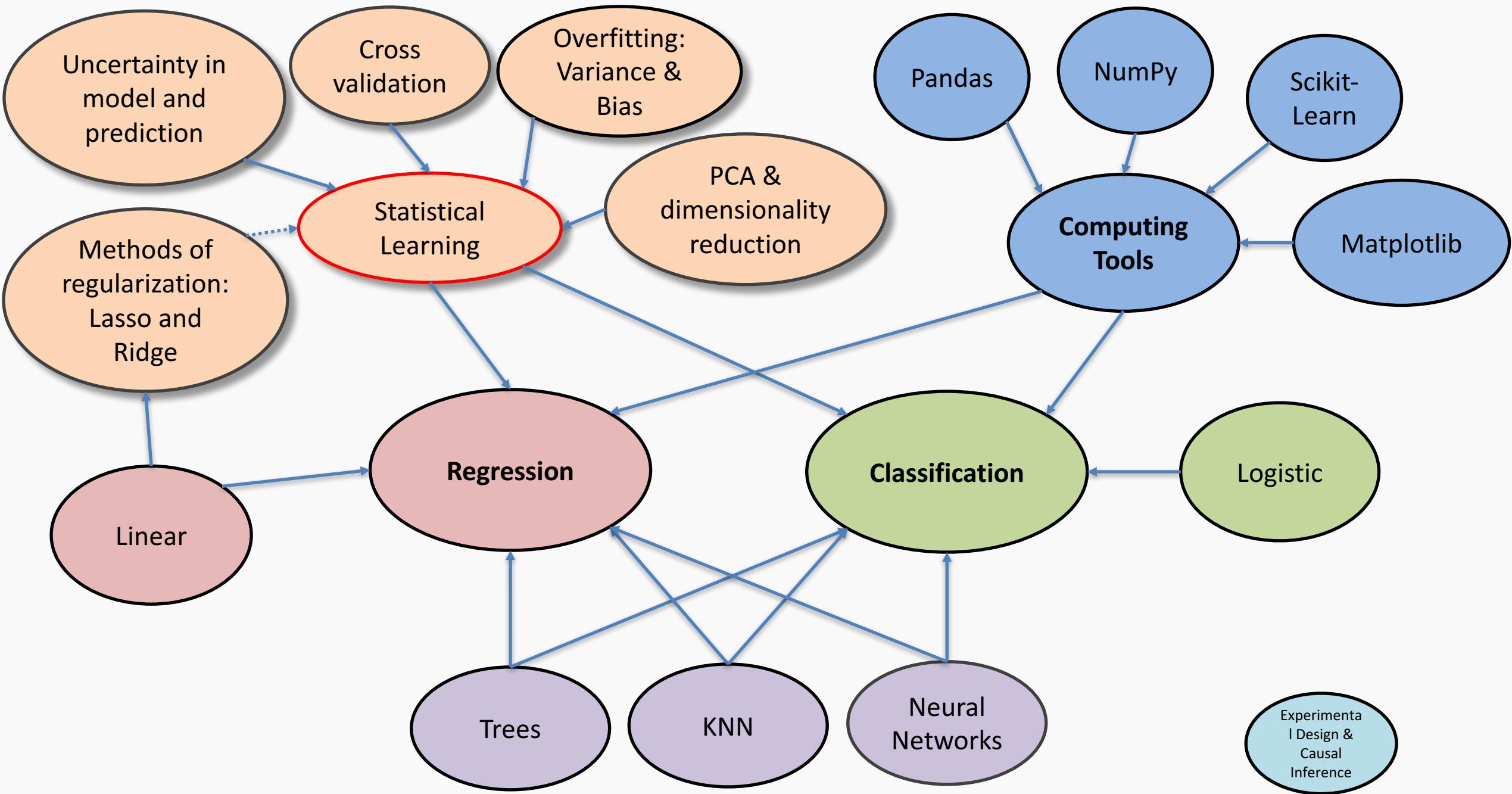
The Math behind PCA

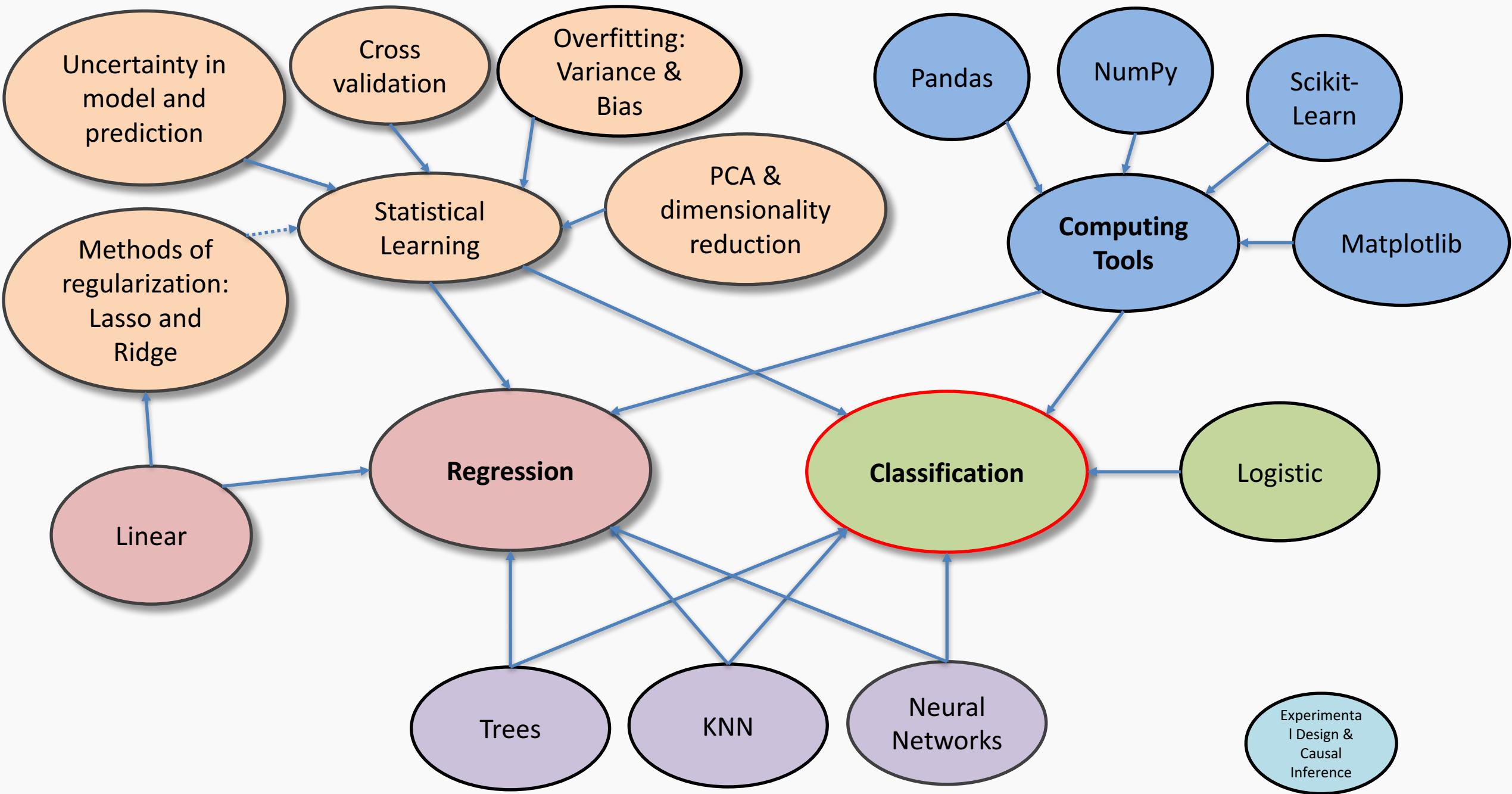
PCA is a well-known result from linear algebra. Let \mathbf{Z} be the $n \times p$ matrix consisting of columns Z_1, \dots, Z_p (the resulting PCA vectors), \mathbf{X} be the $n \times p$ matrix of X_1, \dots, X_p of the original data variables (each standardized to have mean zero and variance one, and without the intercept), and let \mathbf{W} be the $p \times p$ matrix whose columns are the eigenvectors of the square matrix $\mathbf{X}^T \mathbf{X}$, then:

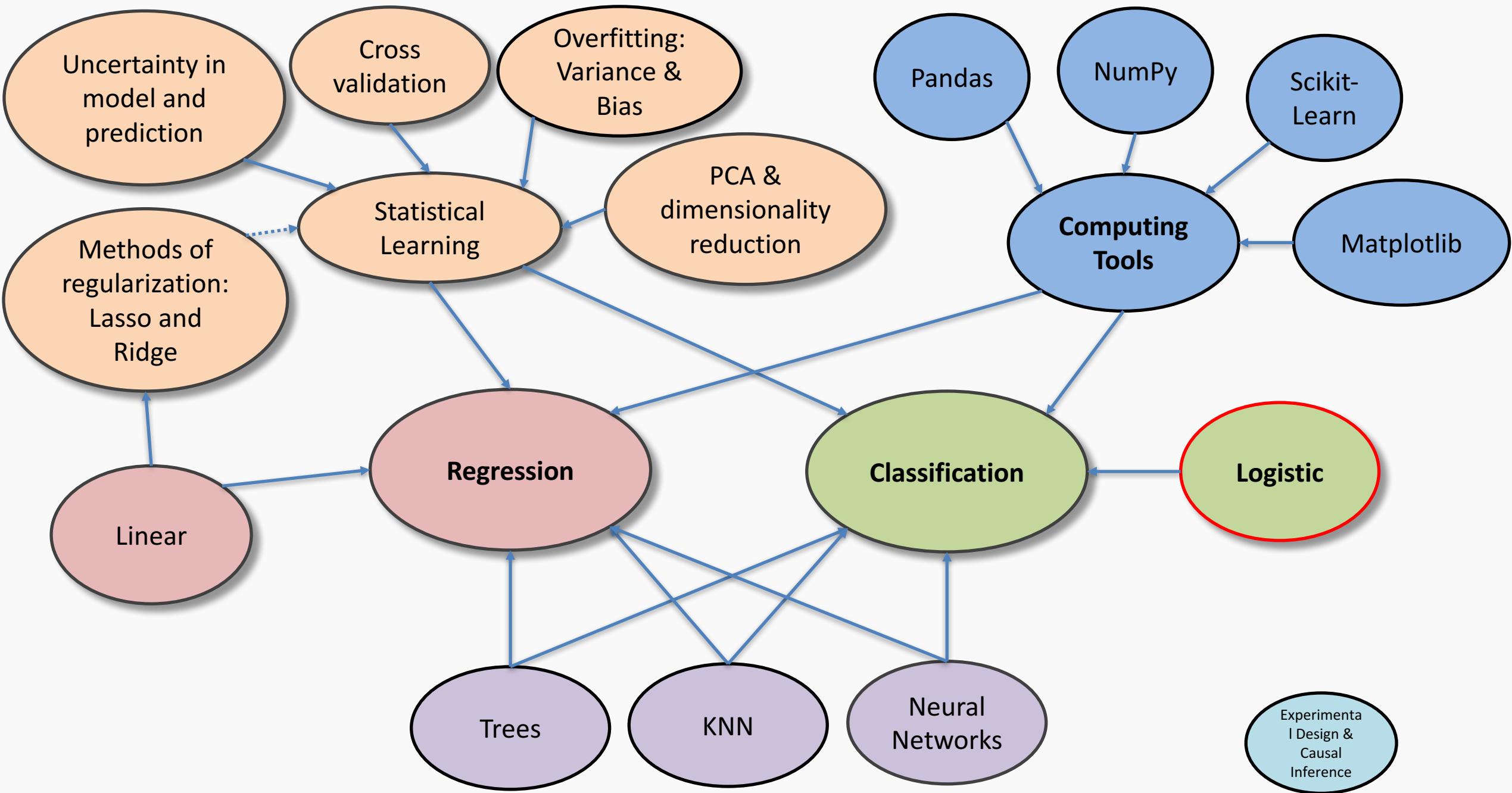
$$\mathbf{Z}_{n \times p} = \mathbf{X}_{n \times p} \mathbf{W}_{p \times p}$$







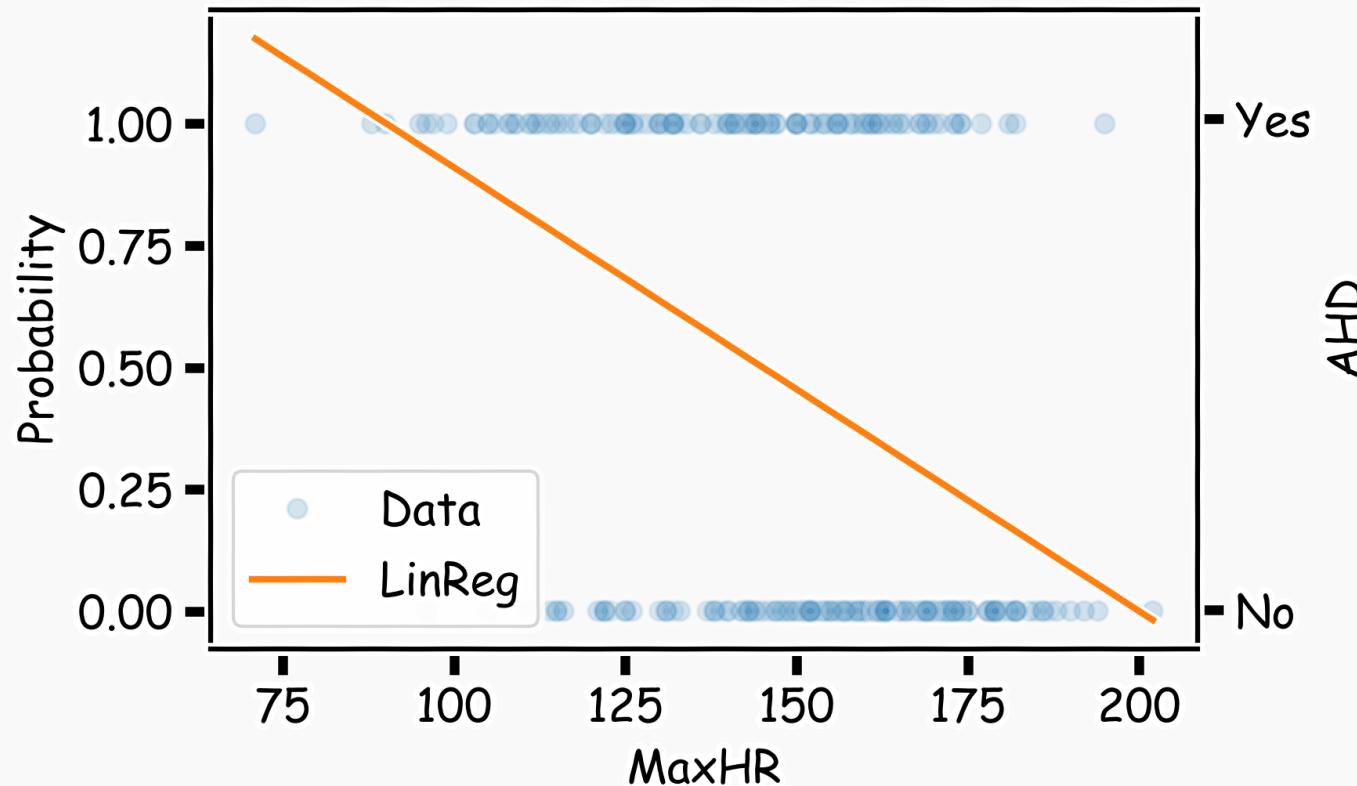




Even Simpler Classification Problem: Binary Response (cont)

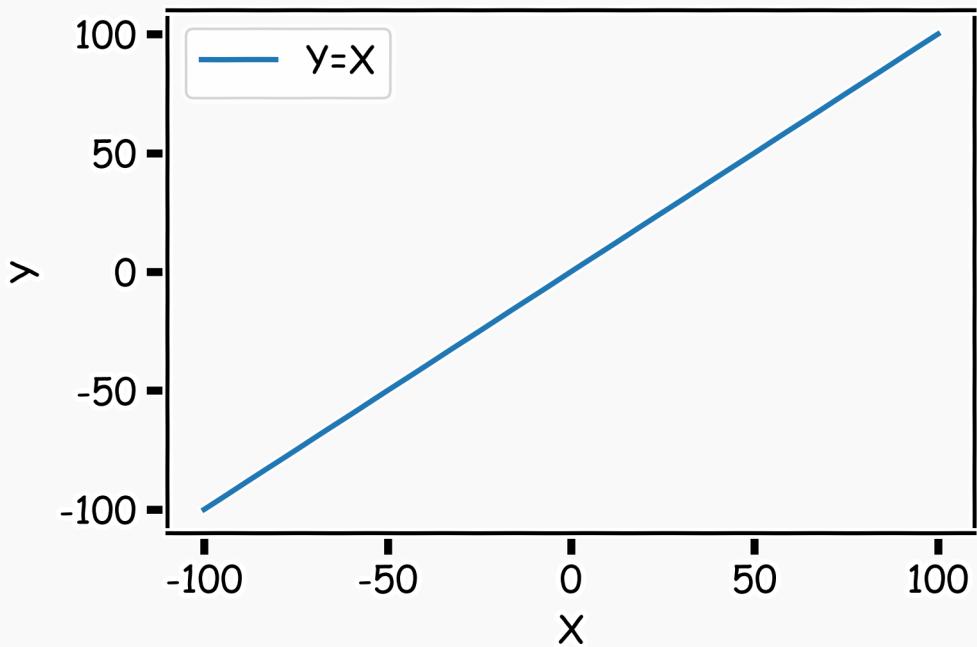
What could go wrong with this linear regression model?

.



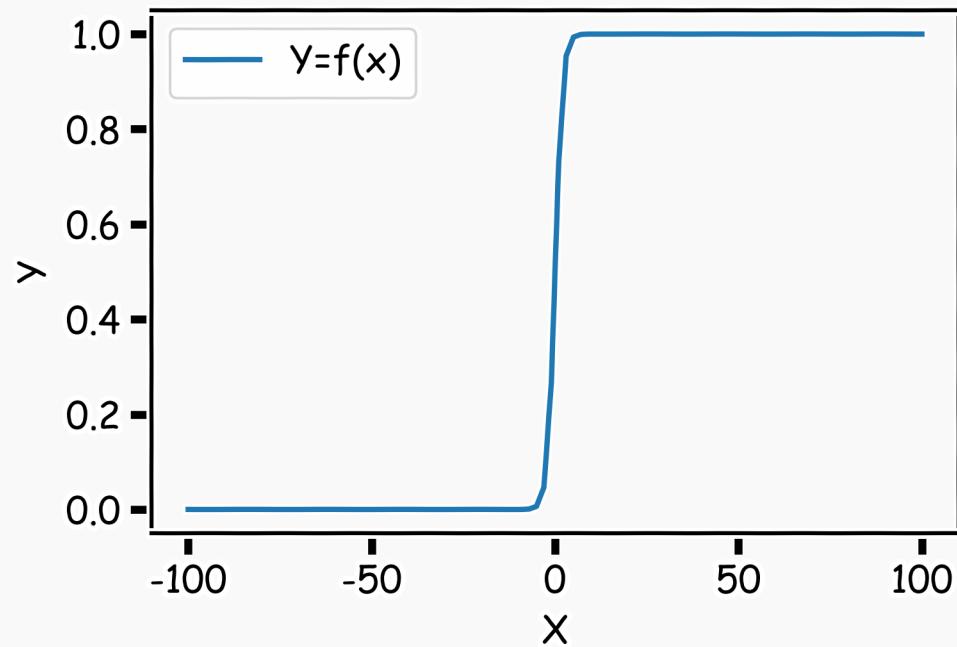
Pavlos Game #45

Think of a function that would do this for us

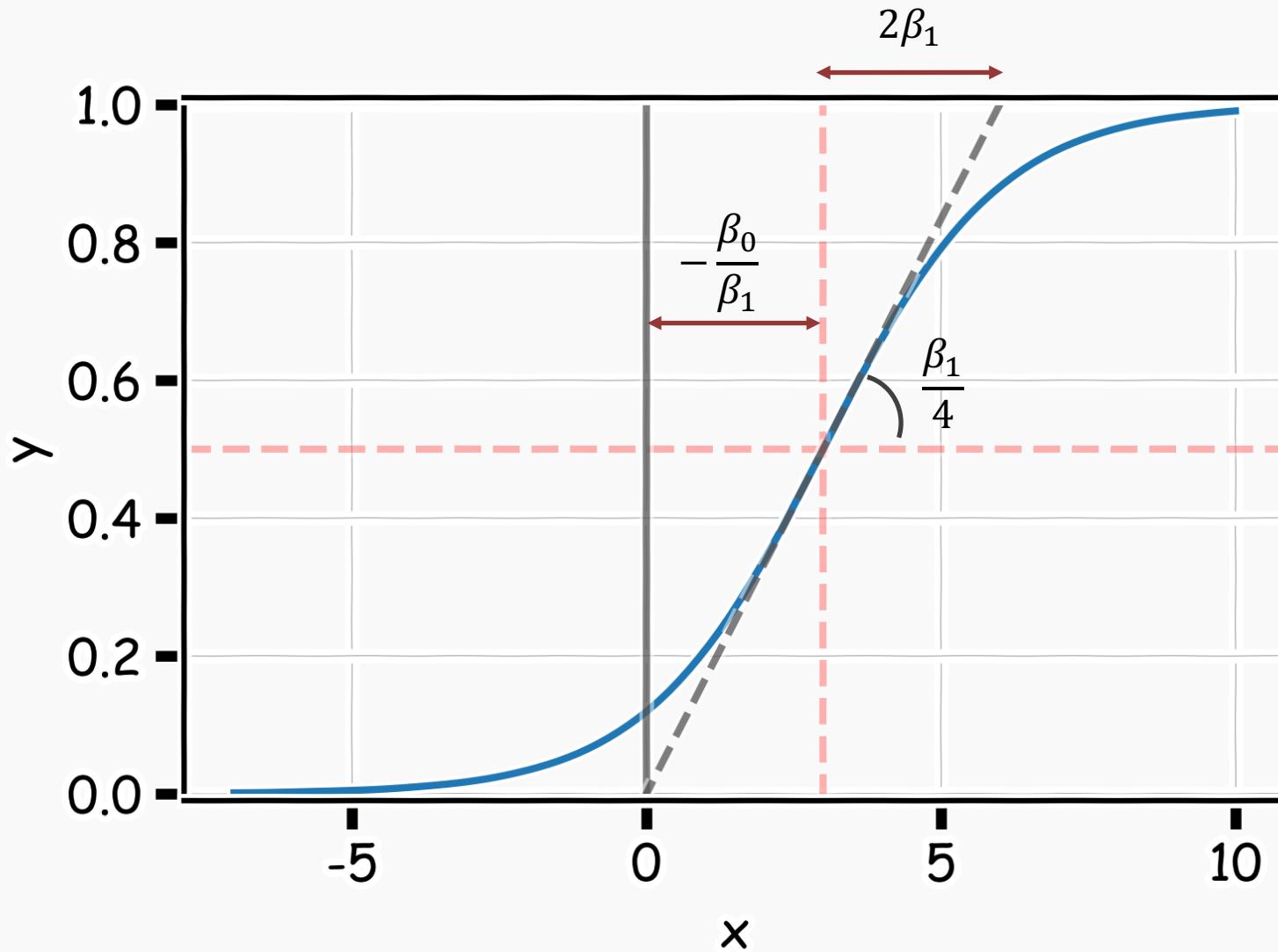


$$Y = f(x)$$

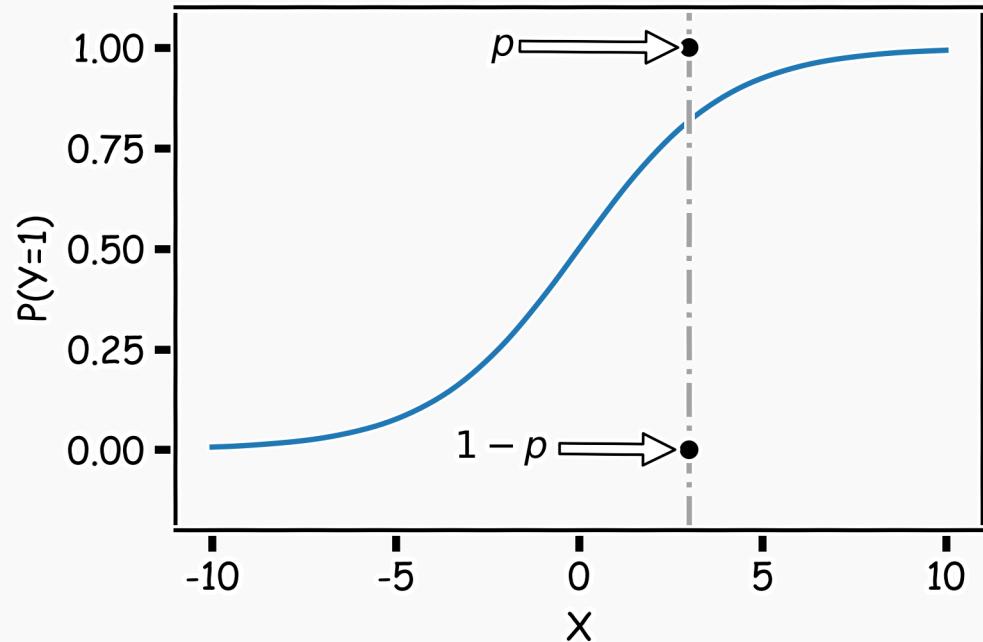
A large grey arrow points from the left graph to the right graph, indicating a transformation or mapping.



Logistic Regression



Estimation in Logistic Regression



Probability $Y = 1$: p

Probability $Y = 0$: $1 - p$

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

where:

$p = P(Y = 1|X = x)$ and therefore p depends on X .

Thus not every p is the same for each individual measurement.

Likelihood

The likelihood of a single observation for p given x and y is:

$$L(p_i | Y_i) = P(Y_i = y_i) = p_i^{y_i} (1 - p_i)^{1-y_i}$$

Given the observations are independent, what is the likelihood function for p ?

$$L(p | Y) = \prod_i P(Y_i = y_i) = \prod_i p_i^{y_i} (1 - p_i)^{1-y_i}$$

$$l(p | Y) = -\log L(p | Y) = -\sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i)$$



Loss Function

$$l(p|Y) = - \sum_i \left[y_i \log \frac{1}{1 + e^{-\beta X_i}} + (1 - y_i) \log \left(1 - \frac{1}{1 + e^{-\beta X_i}} \right) \right]$$

How do we minimize this?

Differentiate, equate to zero and solve for it!

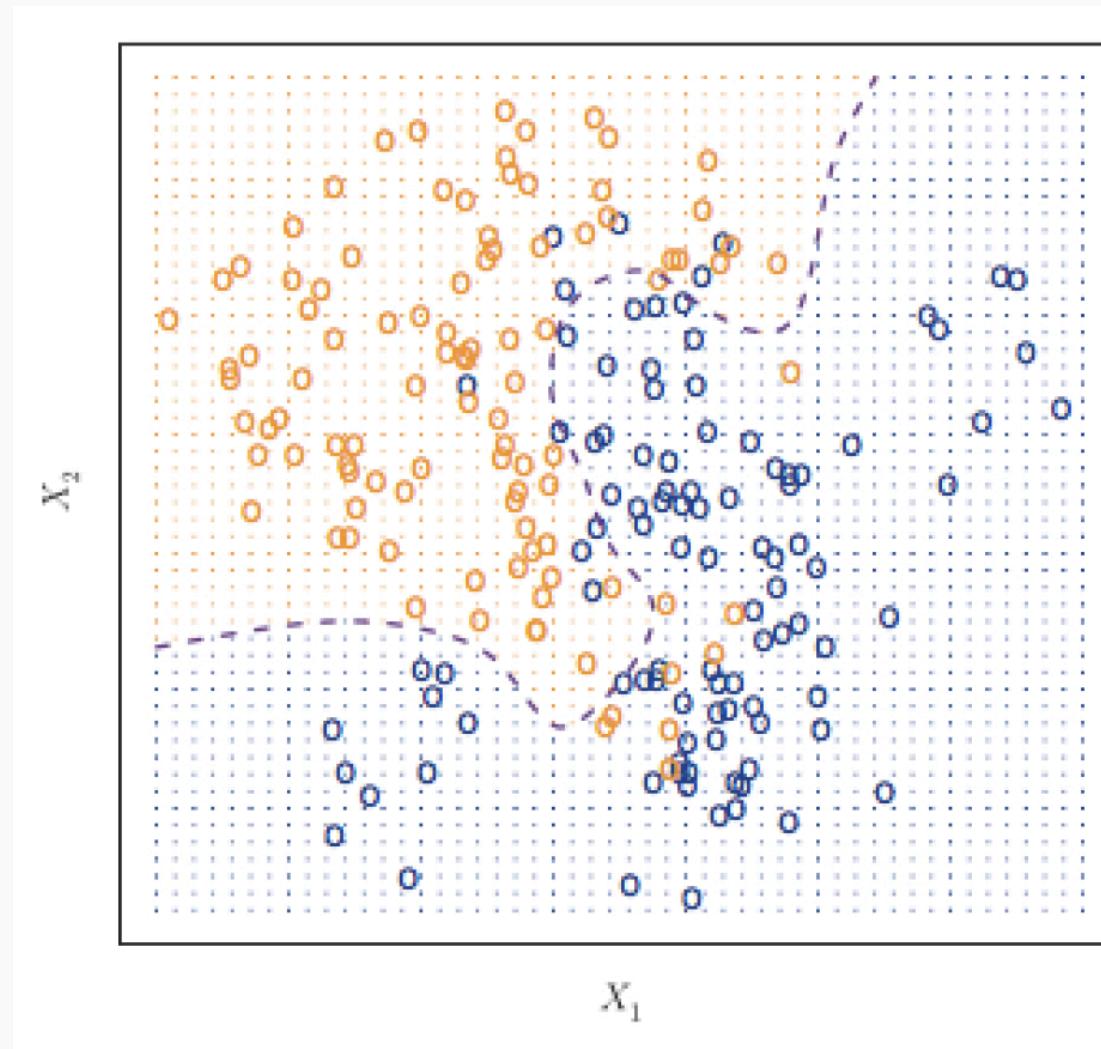
But jeeze does this look messy?! It will not necessarily have a closed form solution.

So how do we determine the parameter estimates? Through an iterative approach (we will talk about this at length in future lectures).



Classifier with two predictors

How can we estimate a classifier, based on logistic regression, for the following plot?



Multiple Logistic Regression

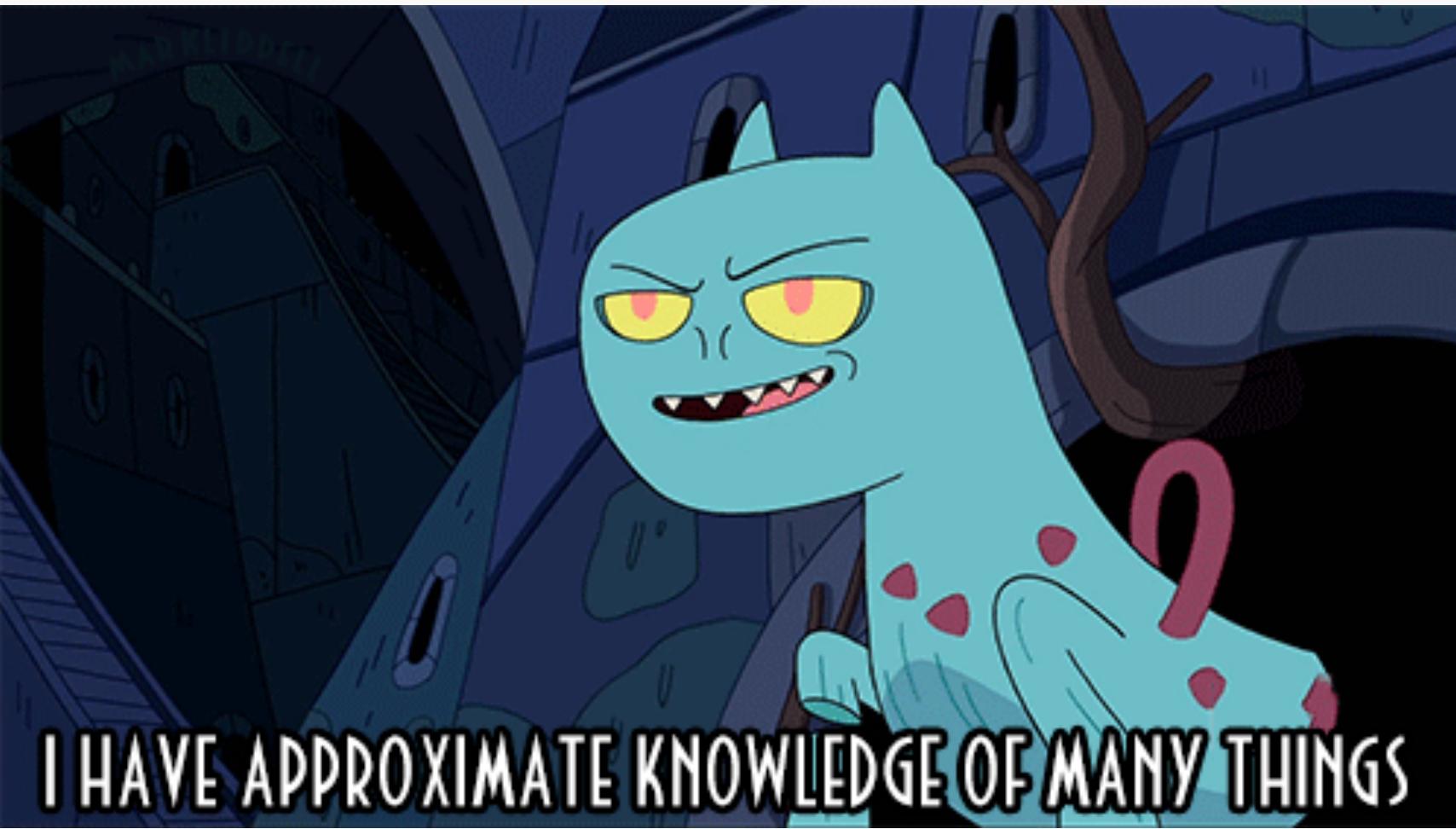
Earlier we saw the general form of simple logistic regression, meaning when there is just one predictor used in the model. What was the model statement (in terms of linear predictors)?

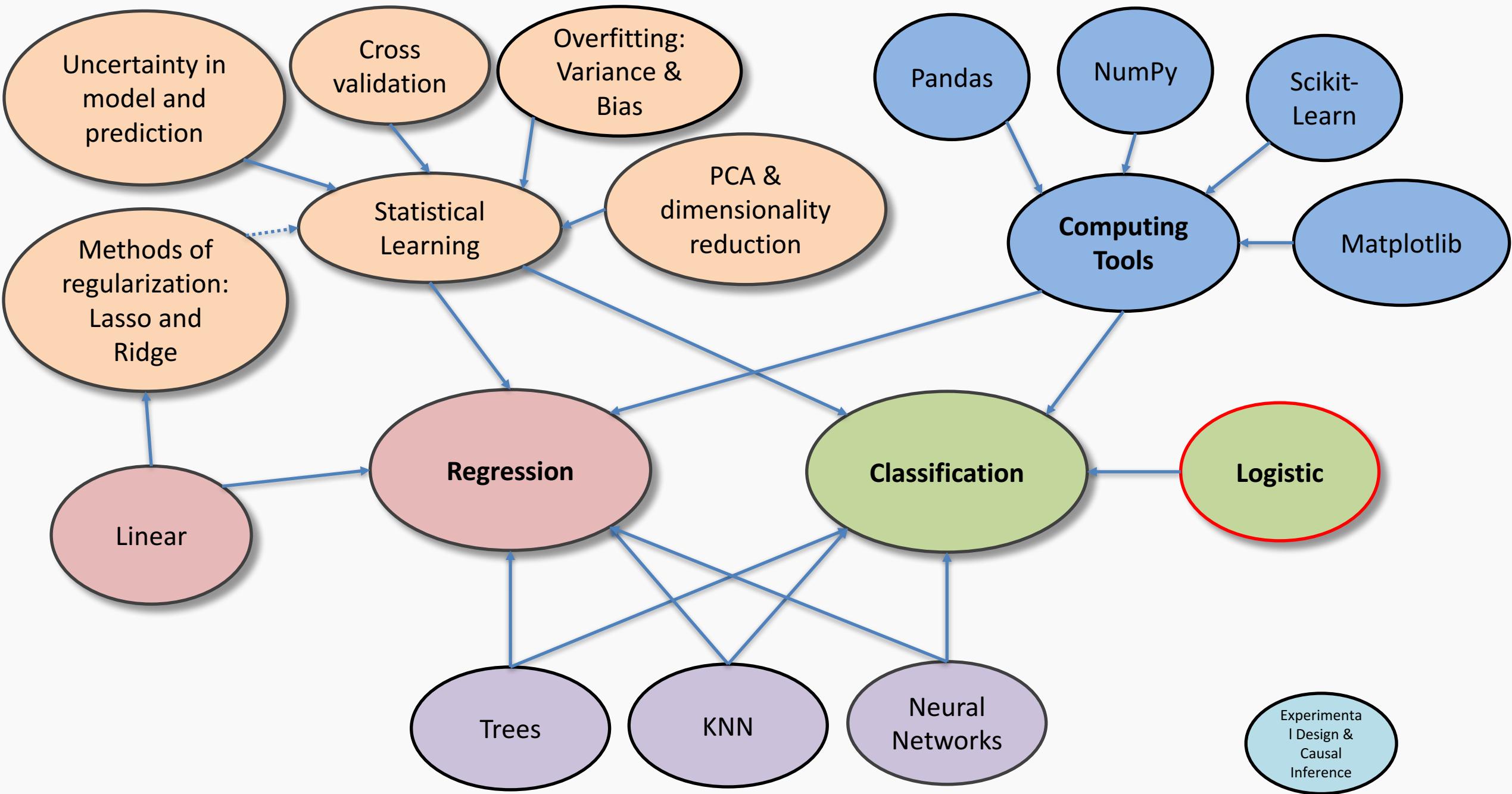
$$\log \left(\frac{P(Y = 1)}{1 - P(Y = 1)} \right) = \beta_0 + \beta_1 X$$

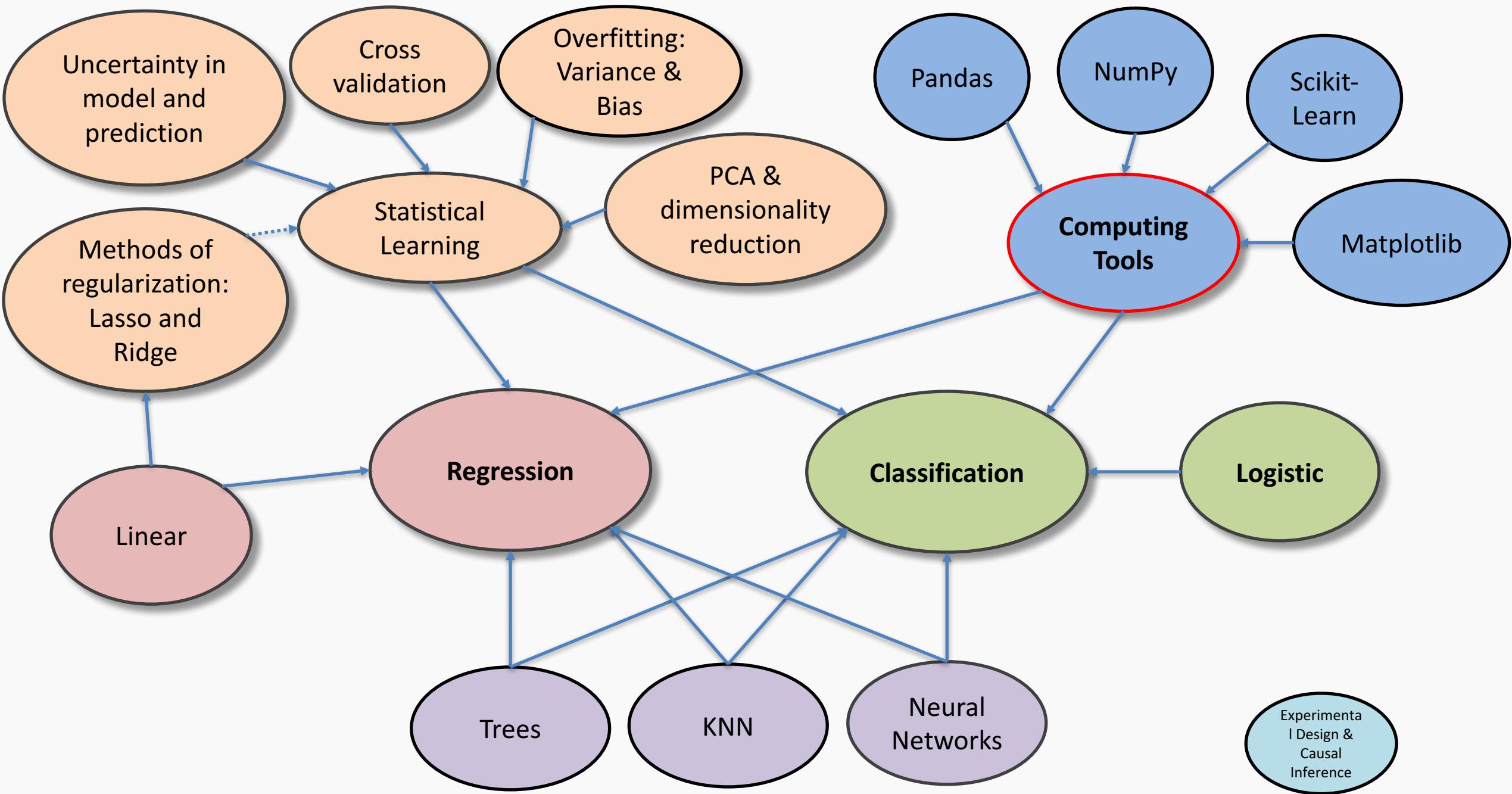
Multiple logistic regression is a generalization to multiple predictors. More specifically we can define a multiple logistic regression model to predict $P(Y = 1)$ as such:

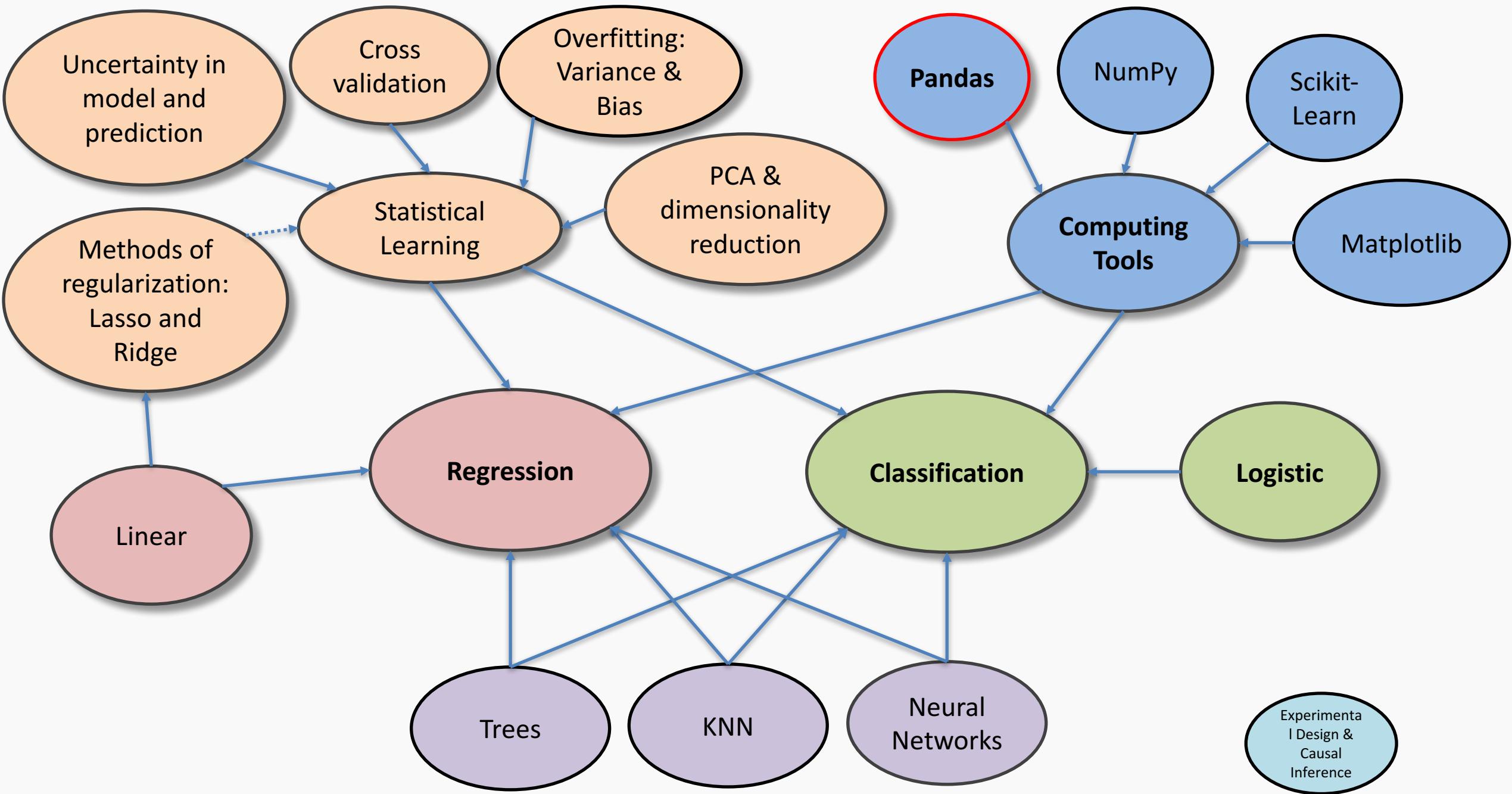
$$\log \left(\frac{P(Y = 1)}{1 - P(Y = 1)} \right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$











Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com



Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

a	3
b	-5
c	7
d	4

Index

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

Columns

Index

A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   >>> 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   >>> 'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
   >>> columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Getting

```
>>> s['b']
-5
>>> df[1]
   Country    Capital  Population
1  India      New Delhi     1303171035
2  Brazil    Brasilia     207847528
```

Also see NumPy Arrays

Get one element
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]
'Belgium'
>>> df.iat[[0], [0]]
'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```

By Label/Position

```
>>> df.ix[2]
   Country    Brazil
   Capital  Brasilia
   Population  207847528
>>> df.ix[:, 'Capital']
0    Brussels
1   New Delhi
2    Brasilia
```

```
>>> df.ix[1, 'Capital']
'New Delhi'
```

Boolean Indexing

```
>>> s[s > 1]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population'] > 12000000000]
```

Setting

```
>>> s['a'] = 6
```

Select single value by row & column
Select single value by row & column labels
Select single row of subset of rows
Select a single column of subset of columns
Select rows and columns

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame
Set index a of Series s to 6

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDF', engine)
```

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns (axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

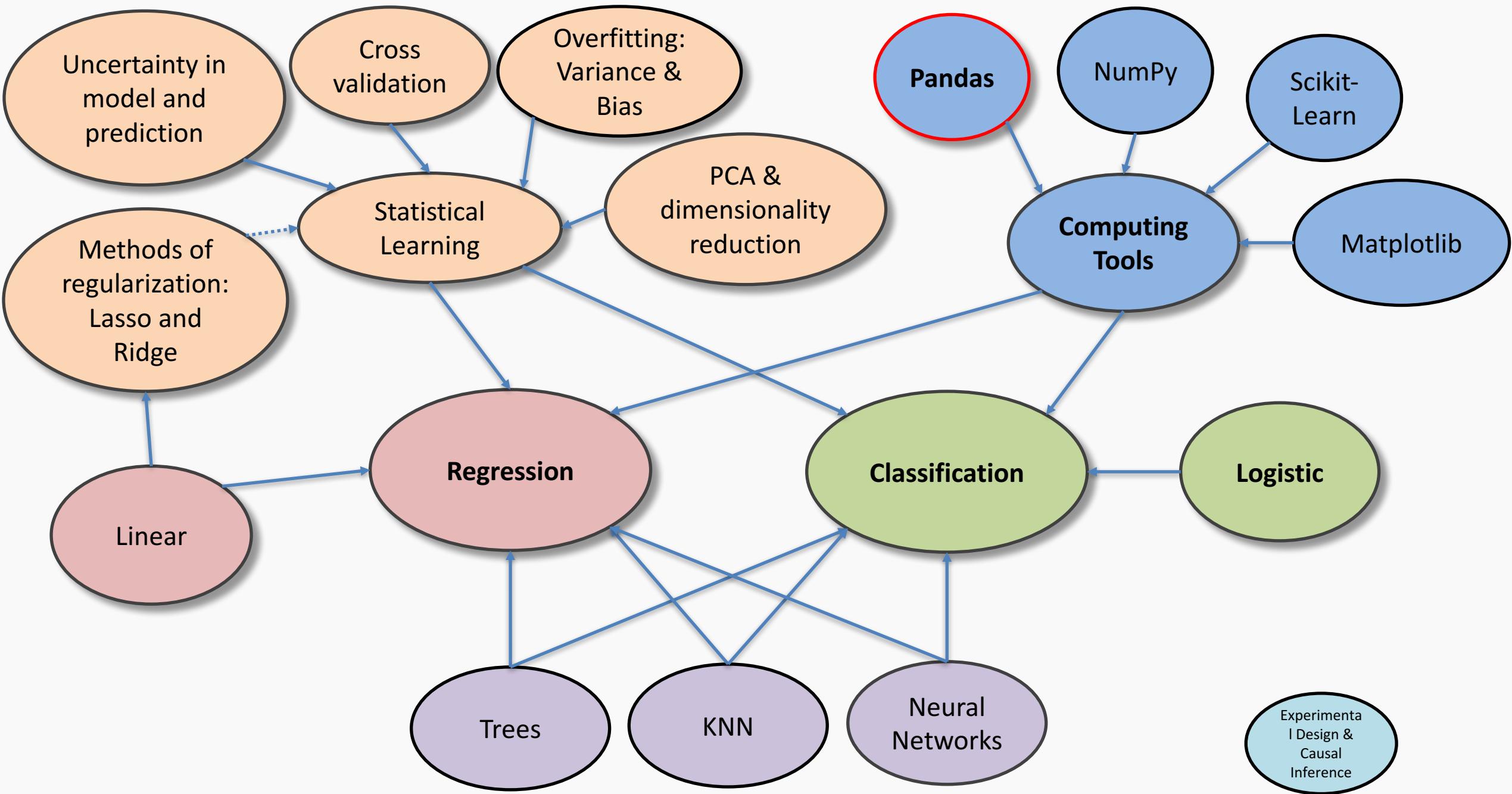
```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a    10.0
b    NaN
c     5.0
d     7.0
```

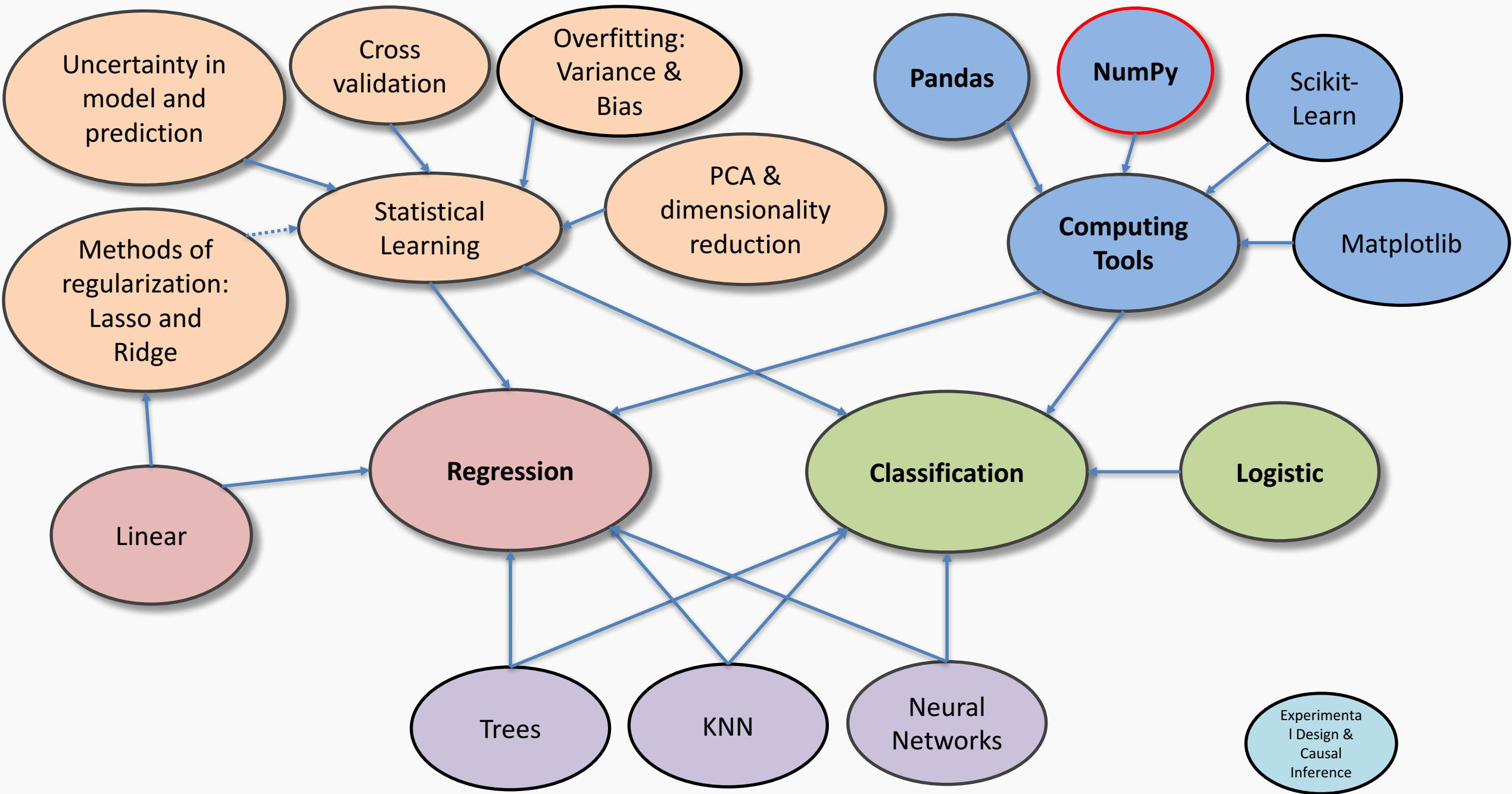
Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c     5.0
d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```







Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com



NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```



NumPy Arrays

1D array

```
[1 2 3]
```

2D array

```
axis 1 → [1.5 2 3  
          4 5 6]  
axis 0 →
```

3D array

```
axis 2 → [axis 1 → [1.5 2 3  
                      4 5 6]  
          axis 0 → [ [1.5 2 3  
                         4 5 6]  
                           [1.5 2 3  
                             4 5 6] ] ] ]
```

Creating Arrays

```
>>> a = np.array([1,2,3])  
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)  
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],  
           dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))  
>>> np.ones((2,3,4),dtype=np.int16)  
>>> d = np.arange(10,25,5)
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of random floats

Inspecting Your Array

```
>>> a.shape  
>>> len(a)  
>>> b.ndim  
>>> e.size  
>>> b.dtype  
>>> b.dtype.name  
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Subsetting

Subsetting
>>> a[2]
3
>>> b[1,2]
6.0

Slicing

```
>>> a[0:2]  
array([1, 2])  
>>> b[0:2,1]  
array([ 2.,  3.])  
>>> b[:1]  
array([[1.5, 2.5, 3.5]])  
>>> c[1,...]  
array([[[ 3.,  4.,  5.],  
       [ 4.,  5.,  6.]]])  
>>> a[ : :-1]  
array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]  
array([1])  
Fancy Indexing  
>>> b[[1, 0, 2]]  
array([ 4.,  1.,  2.])  
>>> b[[1, 0, 2]]  
array([ 4.,  1.,  2.])  
[ 4.,  1.,  2.]
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b  
array([-0.5,  0. ,  0. ],  
      [-3. , -3. , -3. ])  
>>> np.subtract(a,b)  
>>> b + a  
array([[ 2.5,  4. ,  6. ],  
      [ 5. ,  7. ,  9. ]])  
>>> np.add(b,a)  
>>> a / b  
array([[ 0.66666667,  1.        ,  1.        ],  
      [ 0.25       ,  0.4       ,  0.5       ]])  
>>> np.divide(a,b)  
>>> a * b  
array([[ 1.5,  4. ,  9. ],  
      [ 4. , 10. , 18. ]])  
>>> np.multiply(a,b)  
>>> np.exp(b)  
>>> np.sqrt(b)  
>>> np.sin(a)  
>>> np.cos(b)  
>>> np.log(a)  
>>> e.dot(f)  
array([[ 7.,  7.],  
      [ 7.,  7.]])
```

Subtraction

Subtraction
Addition

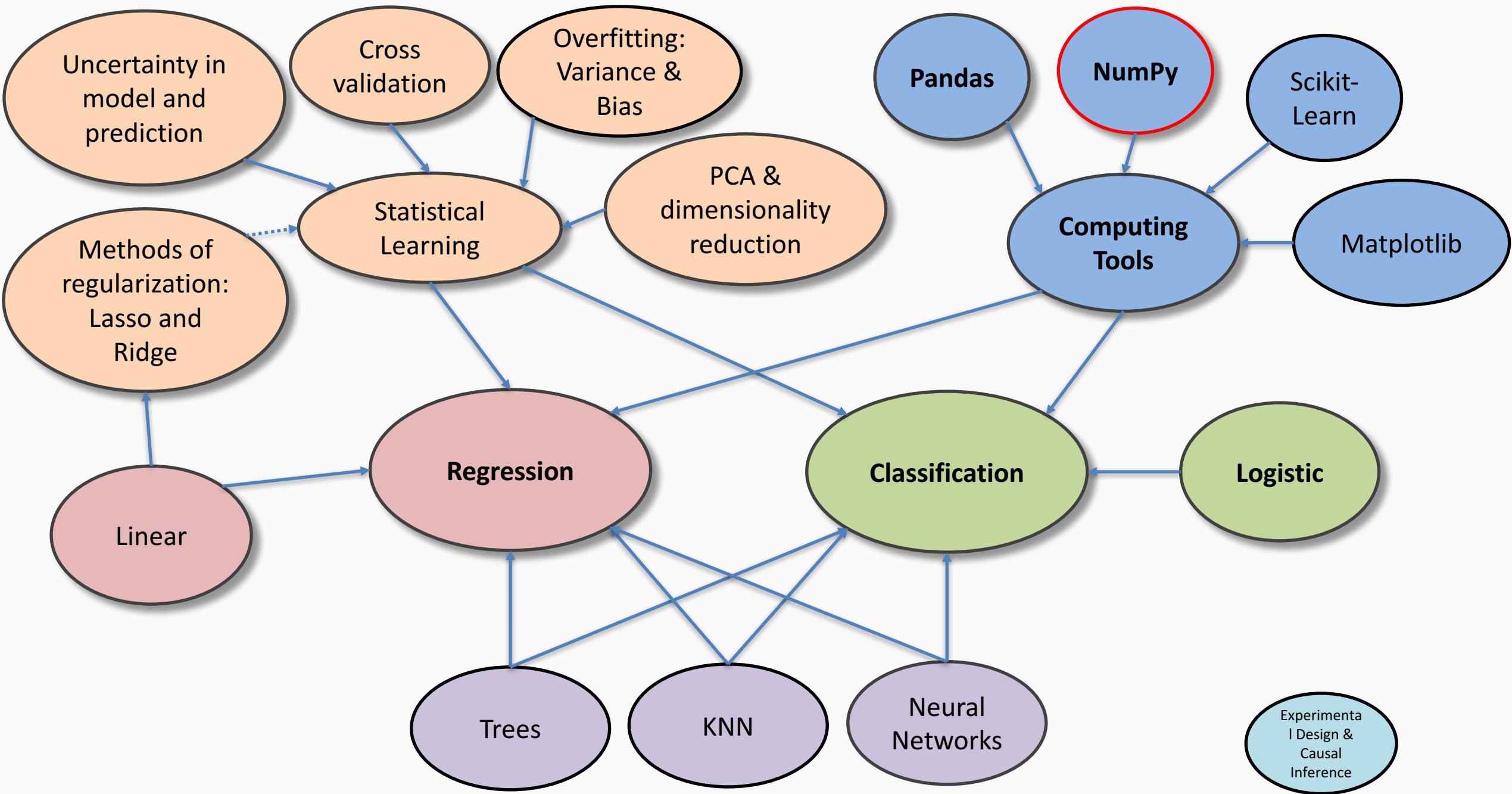
Addition
Division

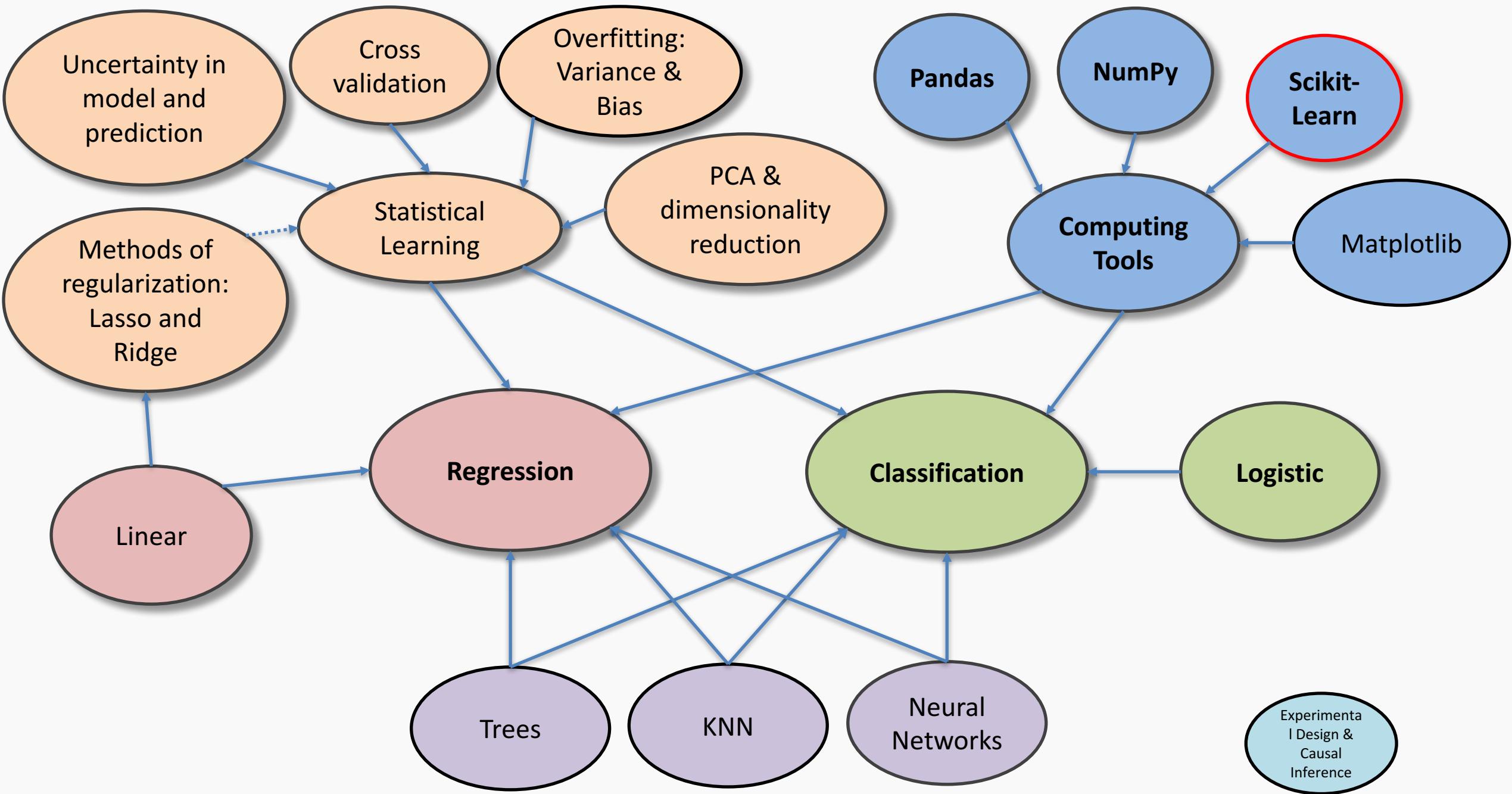
Division
Multiplication

Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Array Manipulation

Transposing
>>> i = np.transpose(j)
>>> i.T
Changing Array Shape
>>> b.ravel()
>>> g.reshape(3,3)
Adding/Removing Elements





Python For Data Science Cheat Sheet

SciPy - Linear Algebra

Learn More Python for Data Science [Interactively](#) at www.datacamp.com



SciPy

The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.



Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[1.5,2,3], (4,5,6), [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:2,0:2]
>>> np.r_[3,[0]*5,-1:1:10j]
>>> np.c_[b,c]
```

Create a dense meshgrid
Create an open meshgrid
Stack arrays vertically (row-wise)
Create stacked column-wise arrays

Shape Manipulation

```
>>> np.transpose(b)
>>> b.flatten()
>>> np.hstack((b,c))
>>> np.vstack((a,b))
>>> np.hsplit(c,2)
>>> np.vsplit(d,2)
```

Permute array dimensions
Flatten the array
Stack arrays horizontally (column-wise)
Stack arrays vertically (row-wise)
Split the array horizontally at the 2nd index
Split the array vertically at the 2nd index

Polynomials

```
>>> from numpy import poly1d
```

Linear Algebra

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains all of the same functions as `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I
>>> linalg.inv(A)
```

Transposition

```
>>> A.T
>>> A.H
```

Trace

```
>>> np.trace(A)
```

Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Rank

```
>>> np.linalg.matrix_rank(C)
```

Determinant

```
>>> linalg.det(A)
```

Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(F,E)
```

Generalized inverse

```
>>> linalg.pinv(C)
```

```
>>> linalg.pinv2(C)
```

Matrix I/O

Addition

```
>>> np.ad
```

Subtraction

```
>>> np.su
```

Division

```
>>> np.di
```

Multiplic

```
>>> A @ D
```

Matrix M

```
>>> np.mu
```

```
>>> np.do
```

```
>>> np.vd
```

```
>>> np.in
```

```
>>> np.ou
```

```
>>> np.te
```

```
>>> np.kr
```

Exponen

```
>>> linal
```

```
>>> linal
```

```
>>> linal
```

Logarith

```
>>> linal
```

Trigonon

```
>>> linal
```

```
>>> linal
```

```
>>> linal
```

Hyperbo

```
>>> linal
```

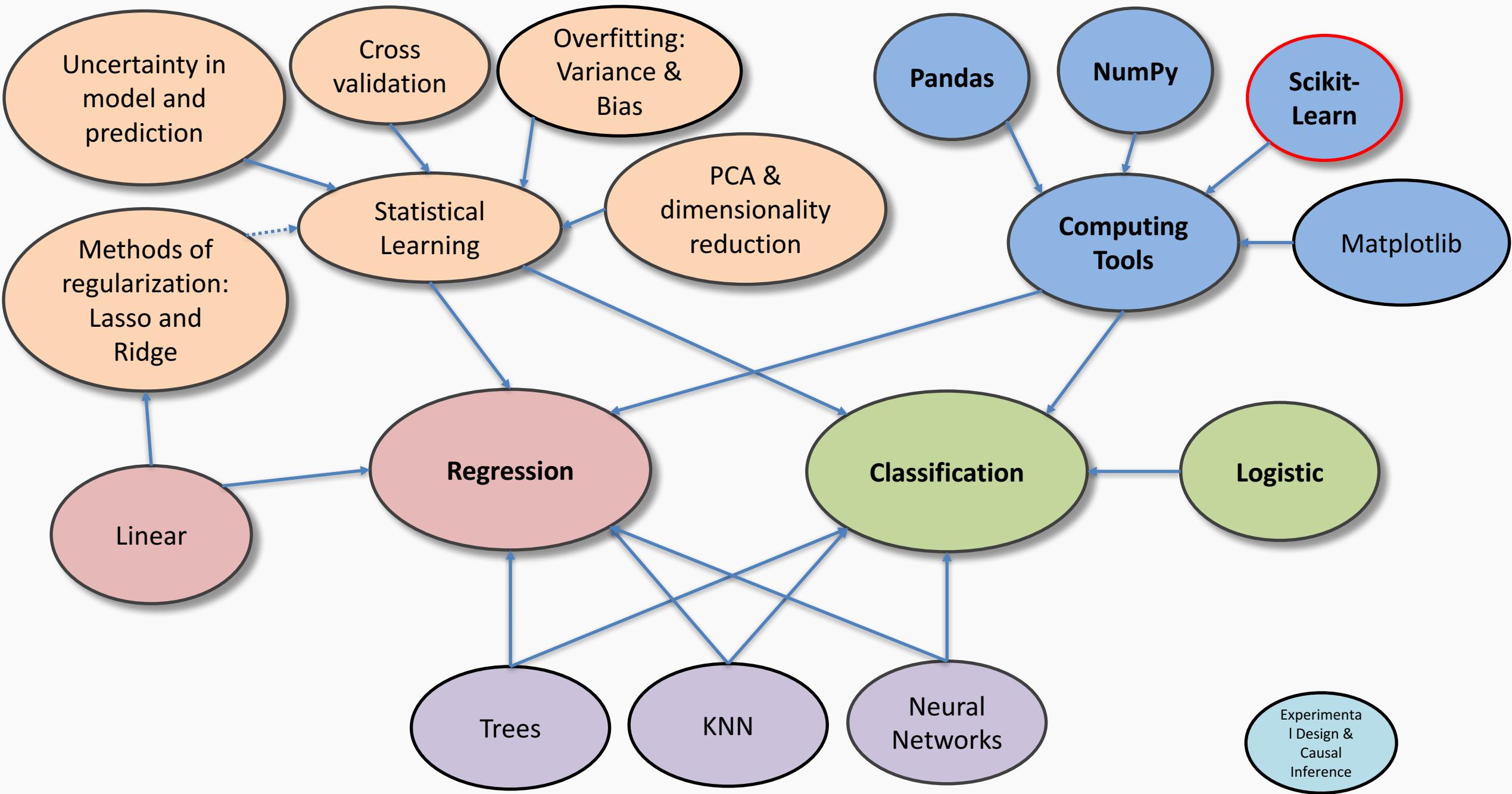
```
>>> linal
```

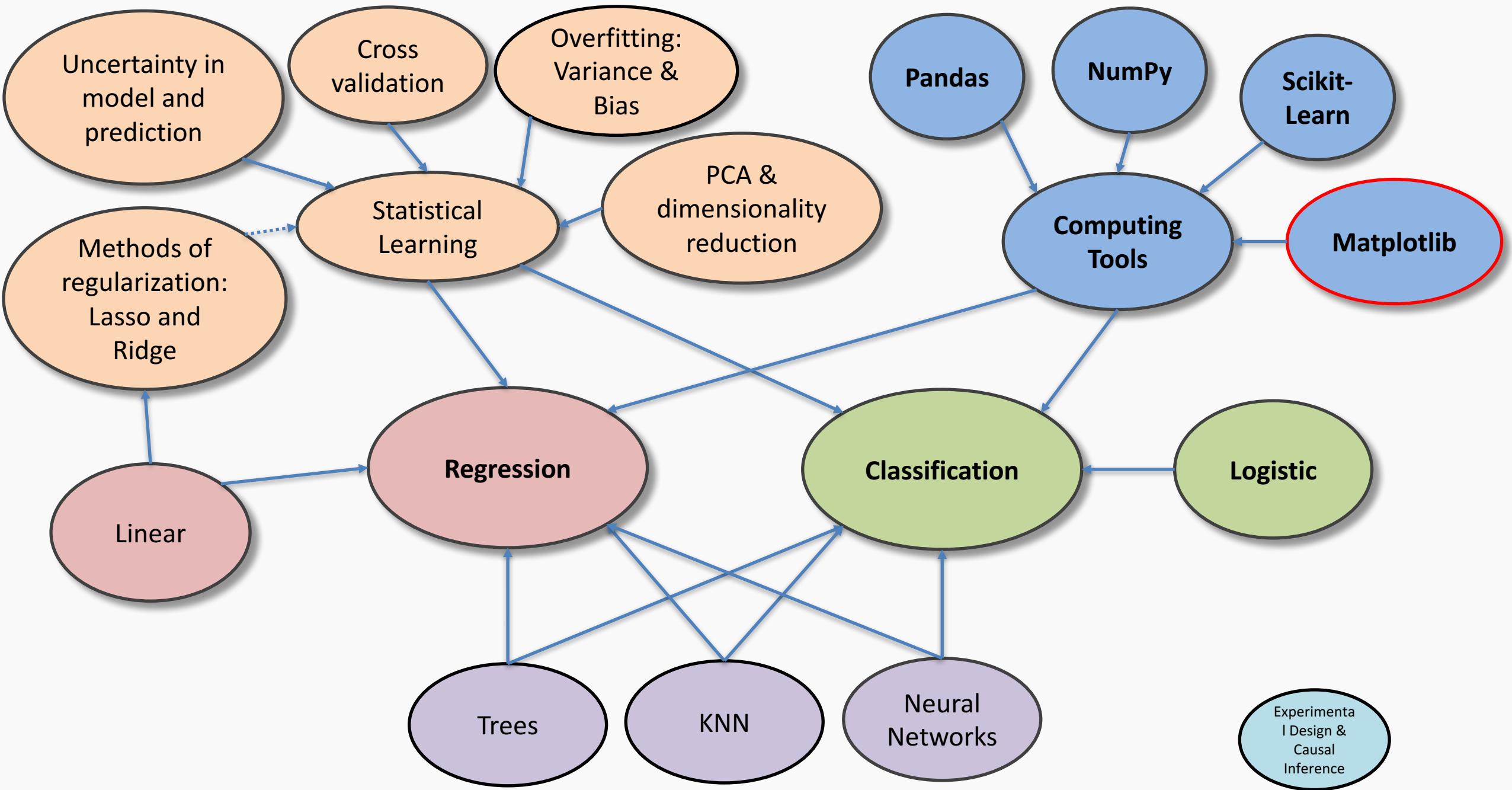
```
>>> linal
```

Matrix S

```
>>> np.si
```

Matrix I





Python For Data Science Cheat Sheet

Matplotlib

Learn Python **Interactively** at www.DataCamp.com



Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.



1 Prepare The Data

Also see [Lists & NumPy](#)

1D Data

```
>>> import numpy as np  
>>> x = np.linspace(0, 10, 100)  
>>> y = np.cos(x)  
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))  
>>> data2 = 3 * np.random.random((10, 10))  
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]  
>>> U = -1 - X**2 + Y  
>>> V = 1 + X - Y**2  
>>> from matplotlib.cbook import get_sample_data  
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

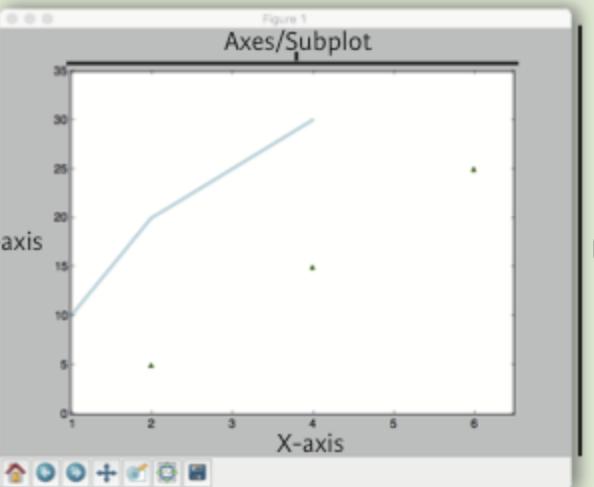
```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()  
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with mat

- 1 Prepare data
- 2 Create plot
- 3 P

```
>>> import matplotlib  
>>> x = [1,2,3,4]  
>>> y = [10,20,25,30]  
>>> fig = plt.figure()  
>>> ax = fig.add_subplot()  
>>> ax.plot(x, y, col  
>>> ax.scatter([2,4,6  
[5,15,  
color=  
marker  
>>> ax.set_xlim(1, 6.  
>>> plt.savefig('foo.p  
>>> plt.show()
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)  
>>> ax.plot(x, y, alpha = 0.4)  
>>> ax.plot(x, y, c='k')  
>>> fig.colorbar(im, orientation='horizontal')  
>>> im = ax.imshow(img,  
cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()  
>>> ax.scatter(x,y,marker=".")  
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)  
>>> plt.plot(x,y,ls='solid')  
>>> plt.plot(x,y,ls='--')  
>>> plt.plot(x,y,'--',x**2,y**2,'-.')  
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

Mathtext

```
>>> plt.title(r'$\sigma_i =
```

Limits, Legends & Layouts

Limits & Autoscaling

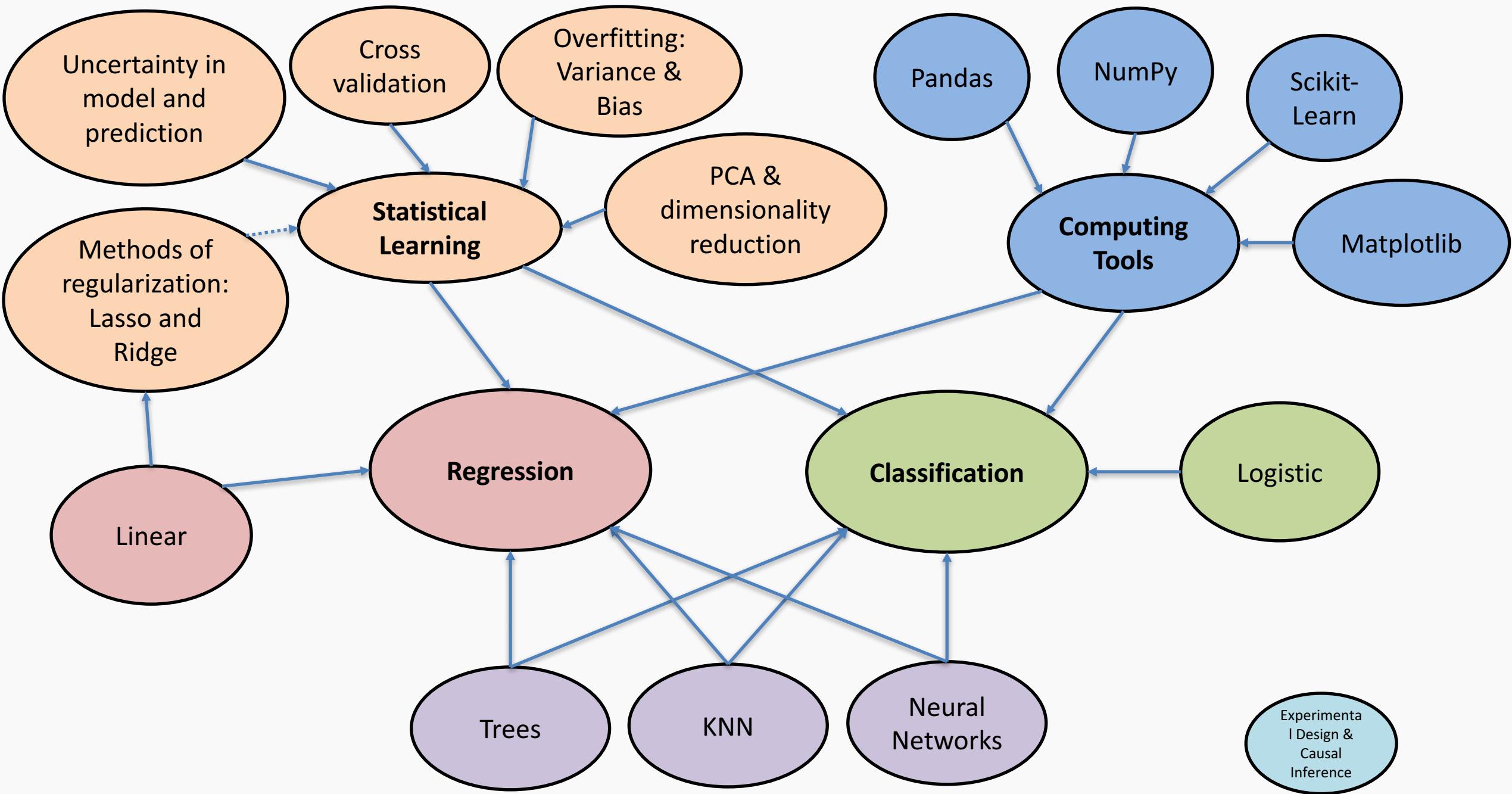
```
>>> ax.margins(x=0.0,y=0.  
>>> ax.axis('equal')  
>>> ax.set(xlim=[0,10.5],y  
>>> ax.set_xlim(0,10.5)
```

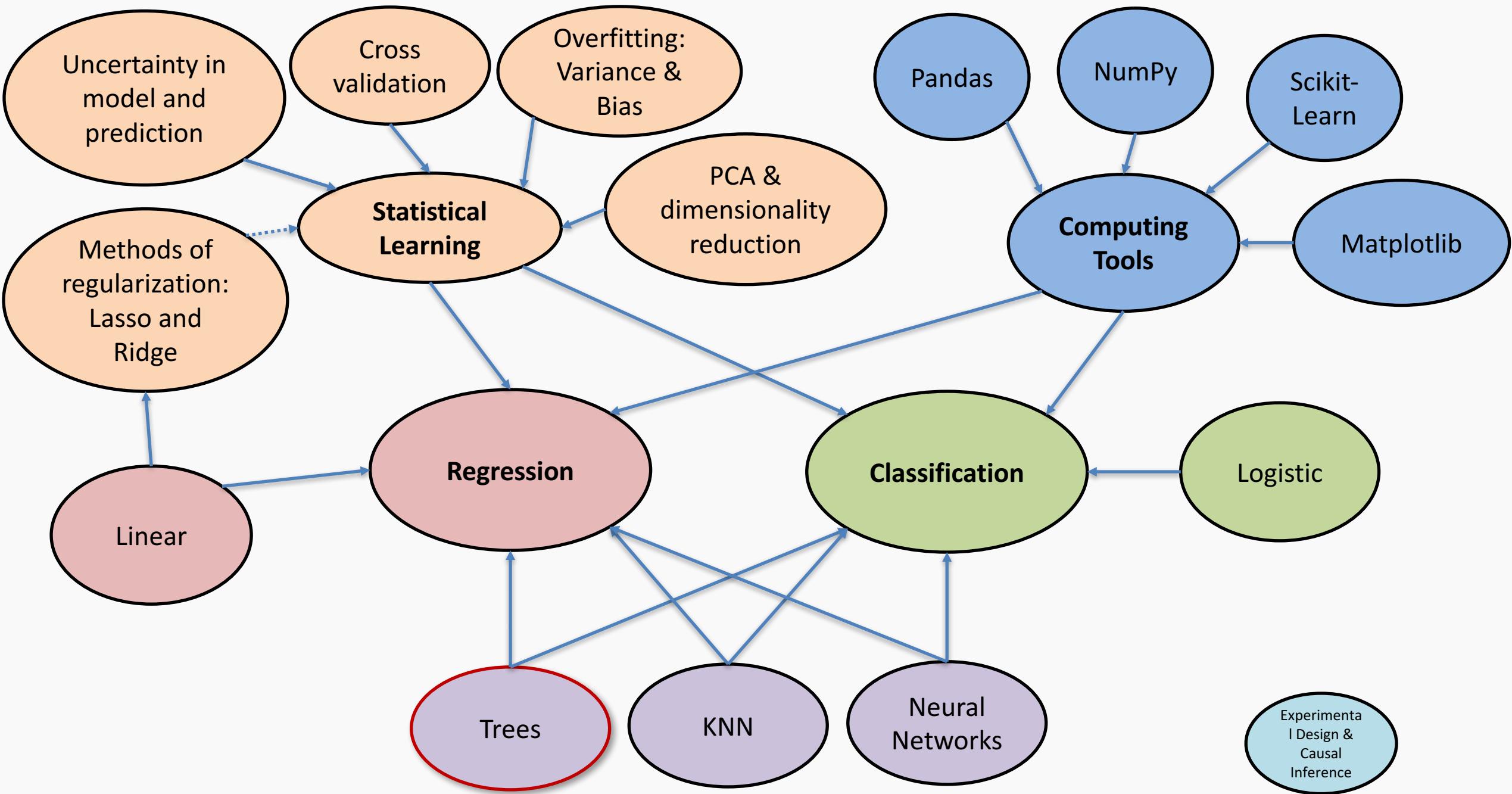
Legends

```
>>> ax.set(title='An Exam  
ylabel='Y-Axis  
xlabel='X-Axis'  
>>> ax.legend(loc='best')
```

Ticks

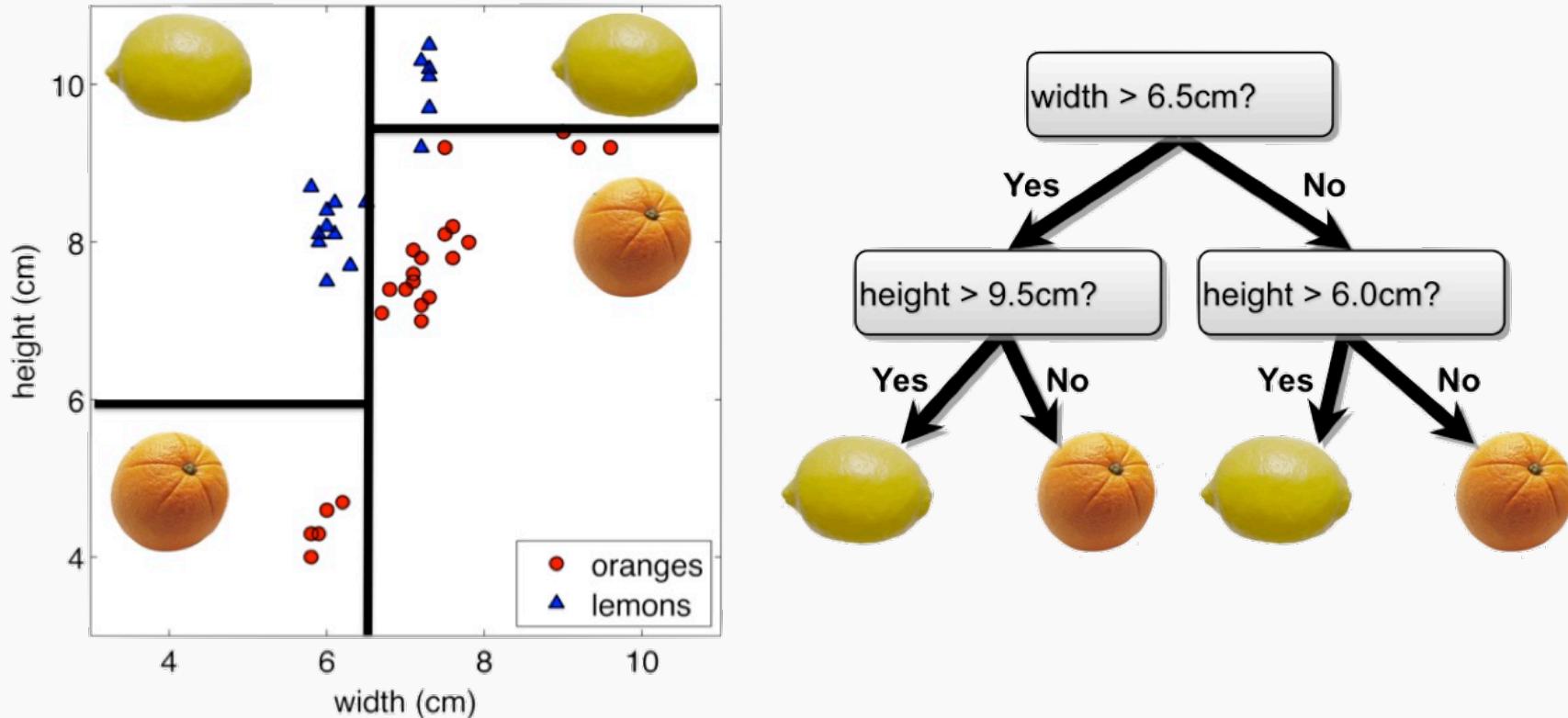
```
>>> ax.xaxis.set(ticks=ra  
ticklabe  
>>> ax.tick_params(axis='  
direct  
length
```



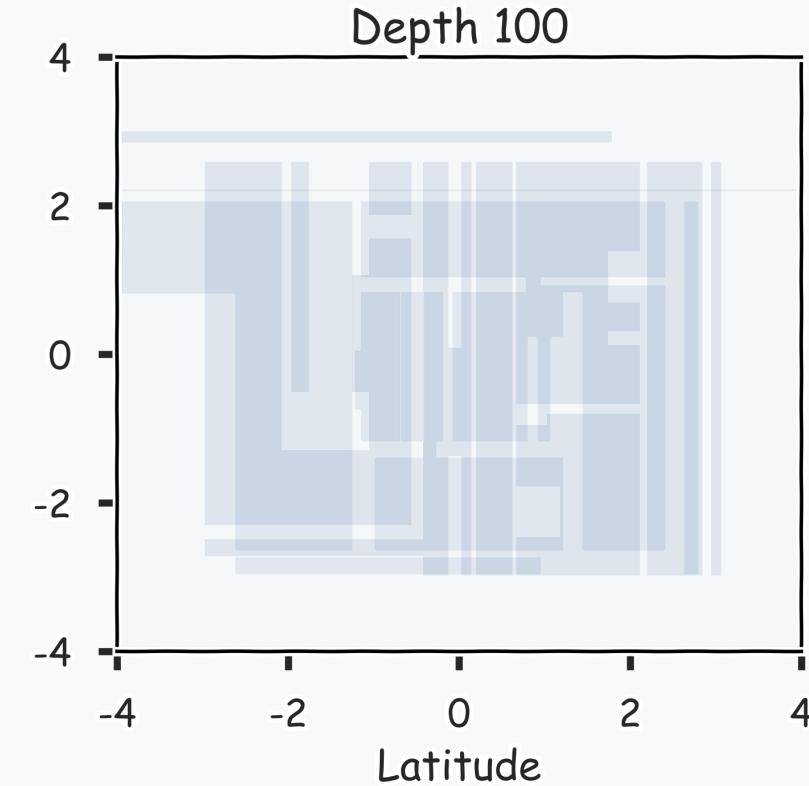
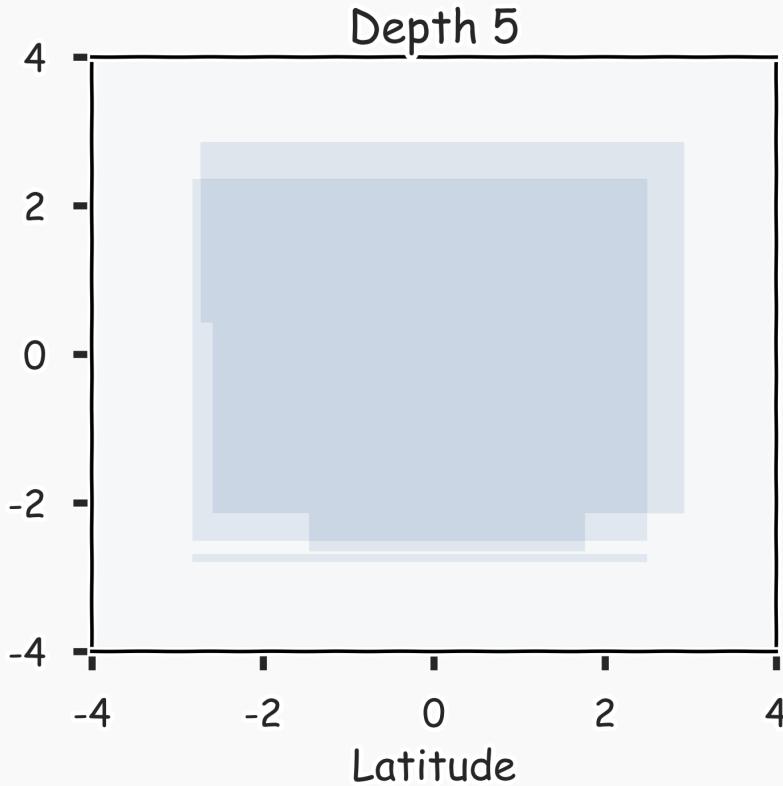
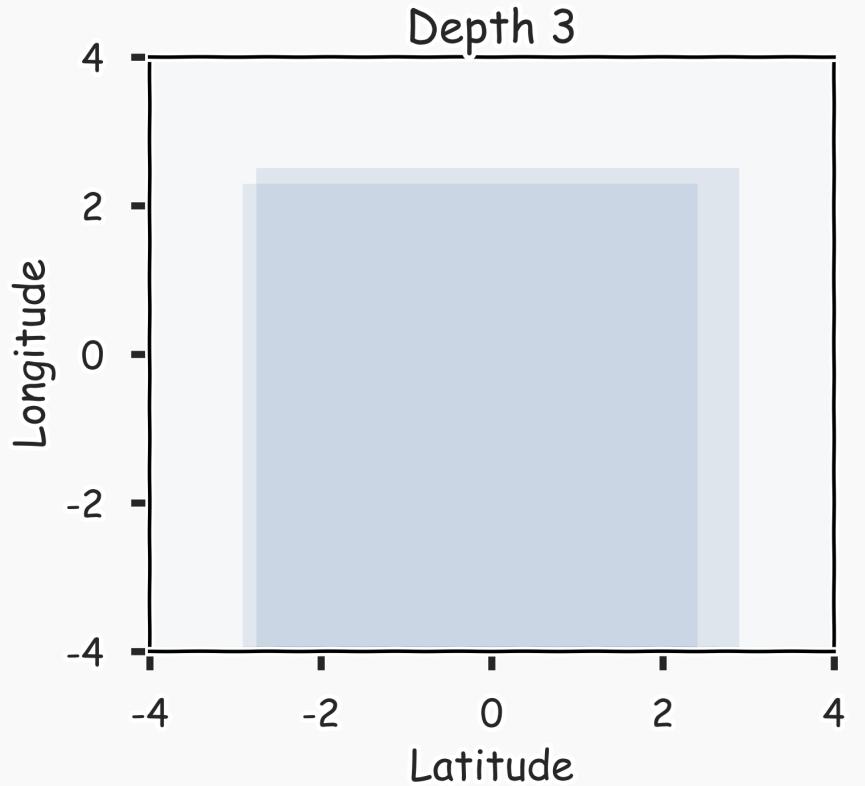


The Geometry of Flow Charts

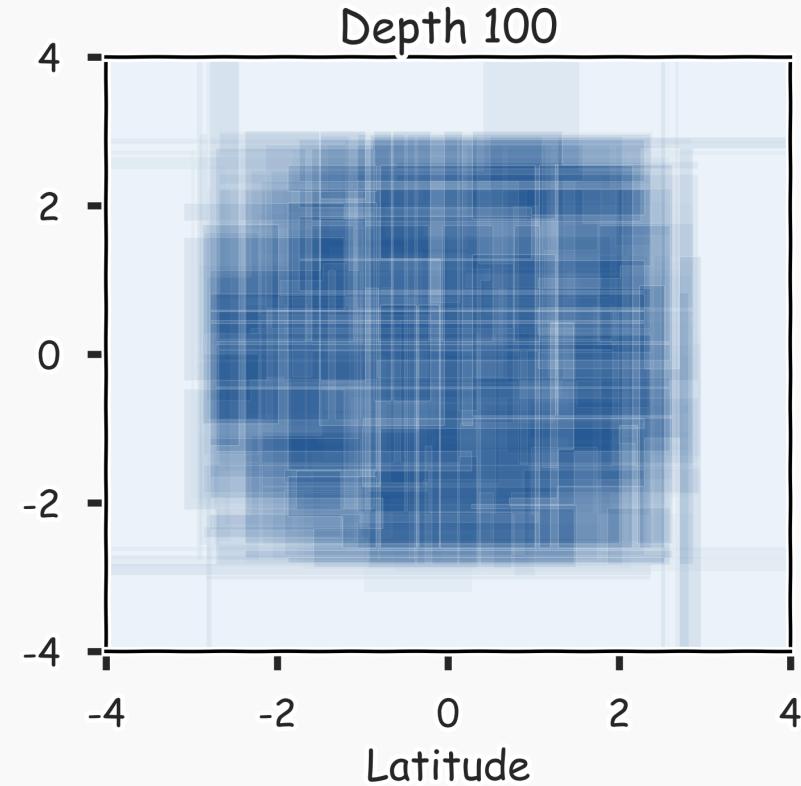
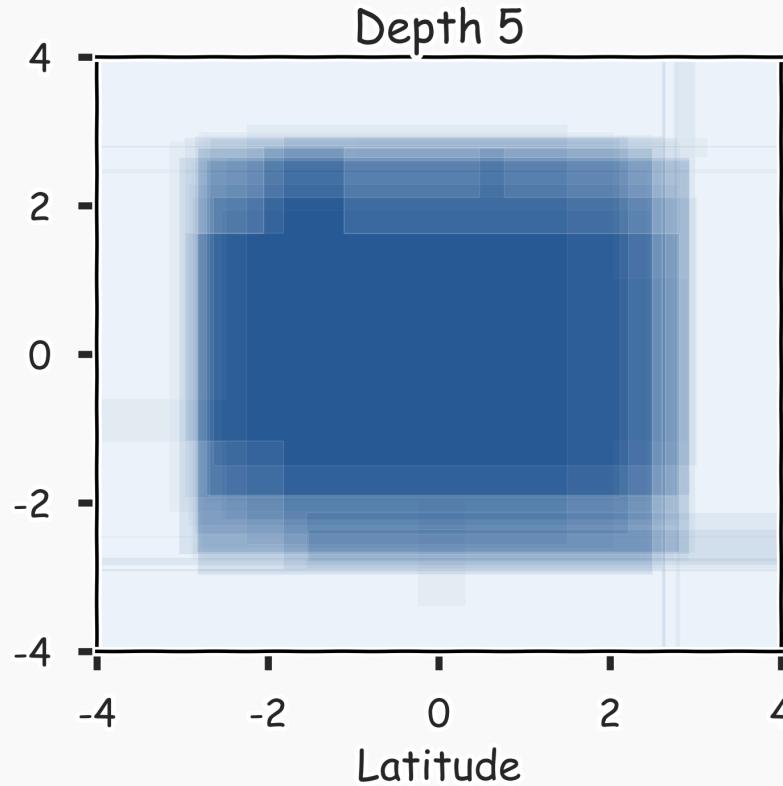
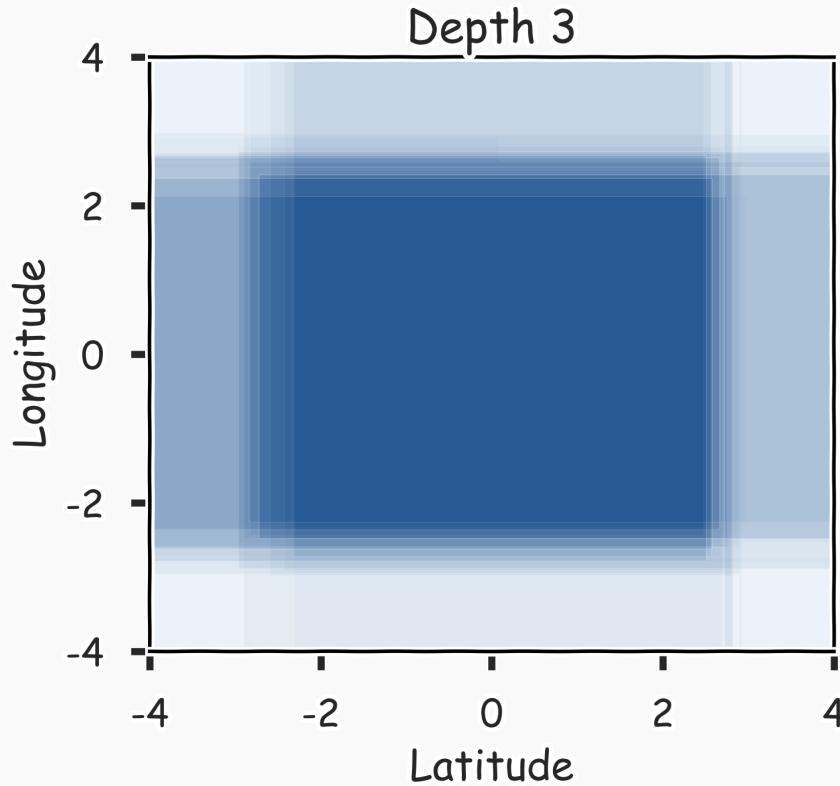
Each comparison and branching represents splitting a region in the feature space on a single feature. Typically, at each iteration, we split once along one dimension (one predictor). Why?



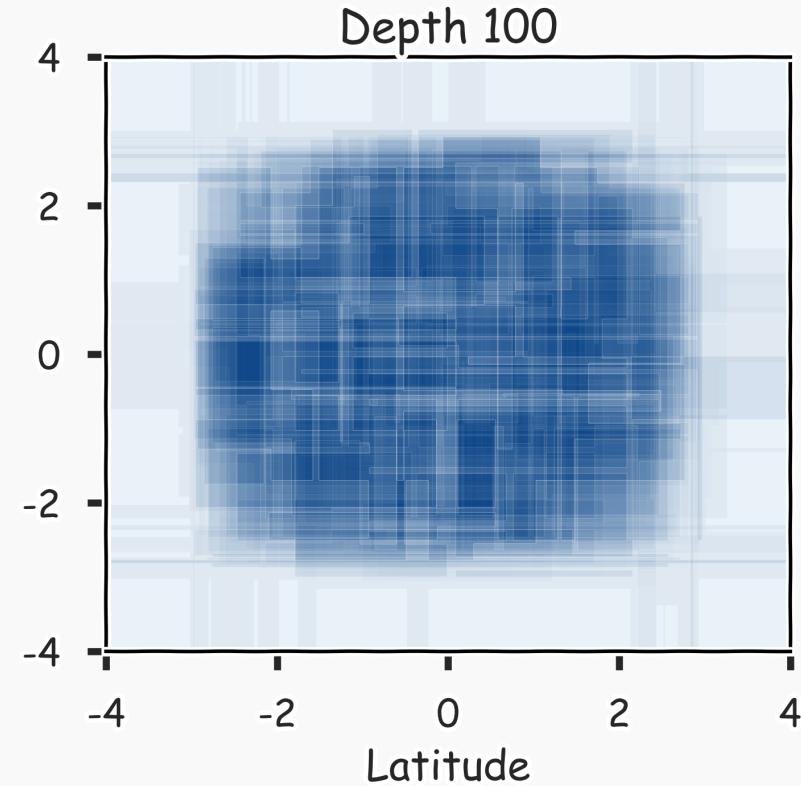
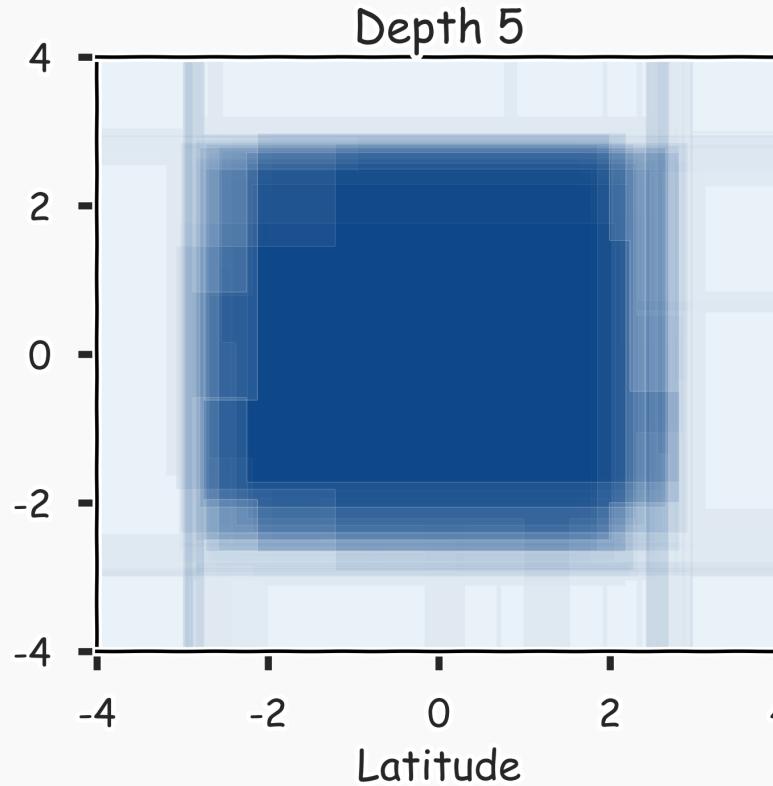
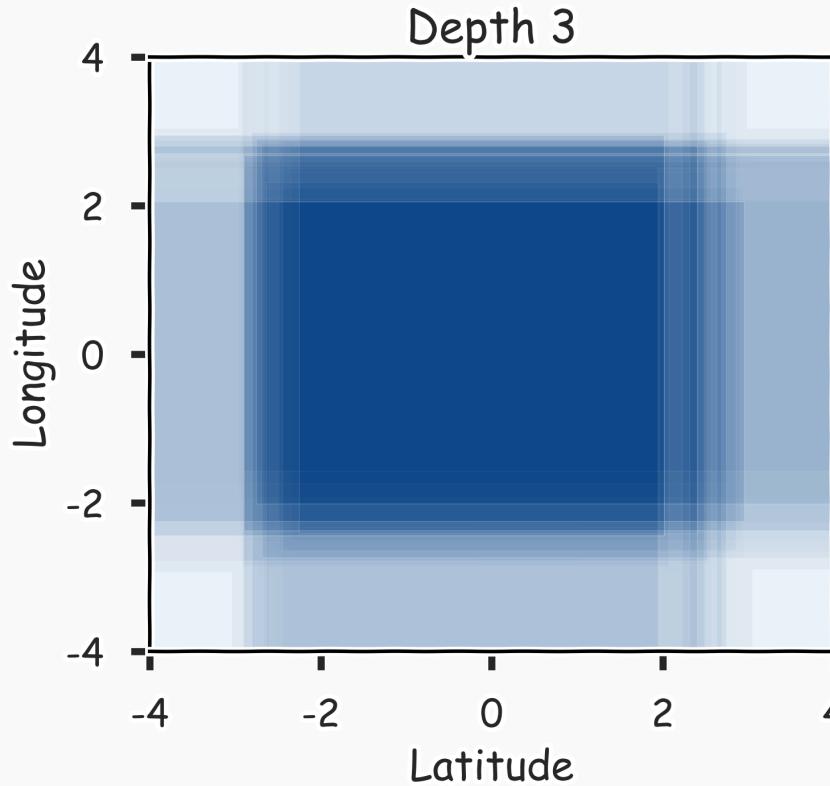
Combine them? 2 magic realisms



Combine them? 20 magic realisms



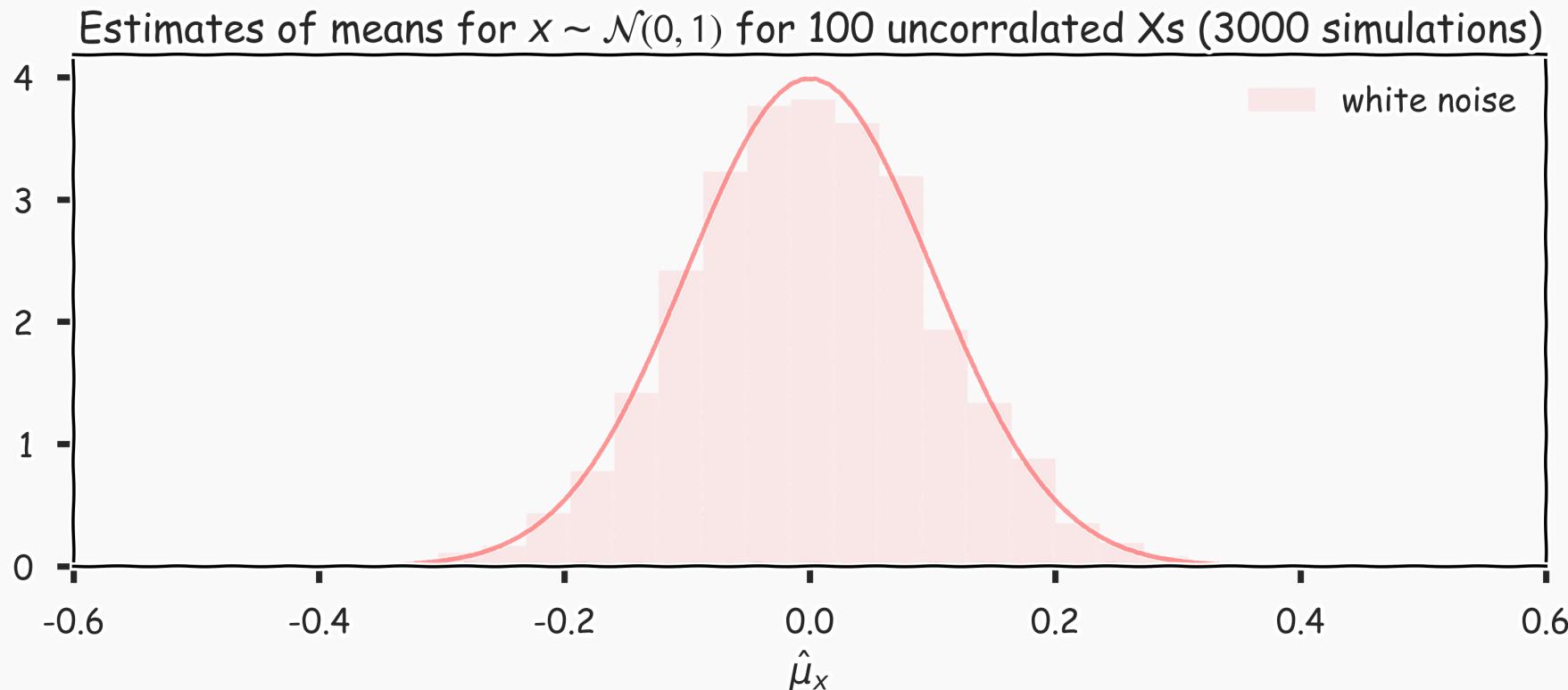
Combine them? 100 magic realisms



Improving on Bagging

Recall, for B number of identically and independently distributed variable, X , with variance σ^2 , the variance of the estimate of the mean is :

$$\text{var}(\hat{\mu}_x) = \frac{\sigma^2}{B}$$

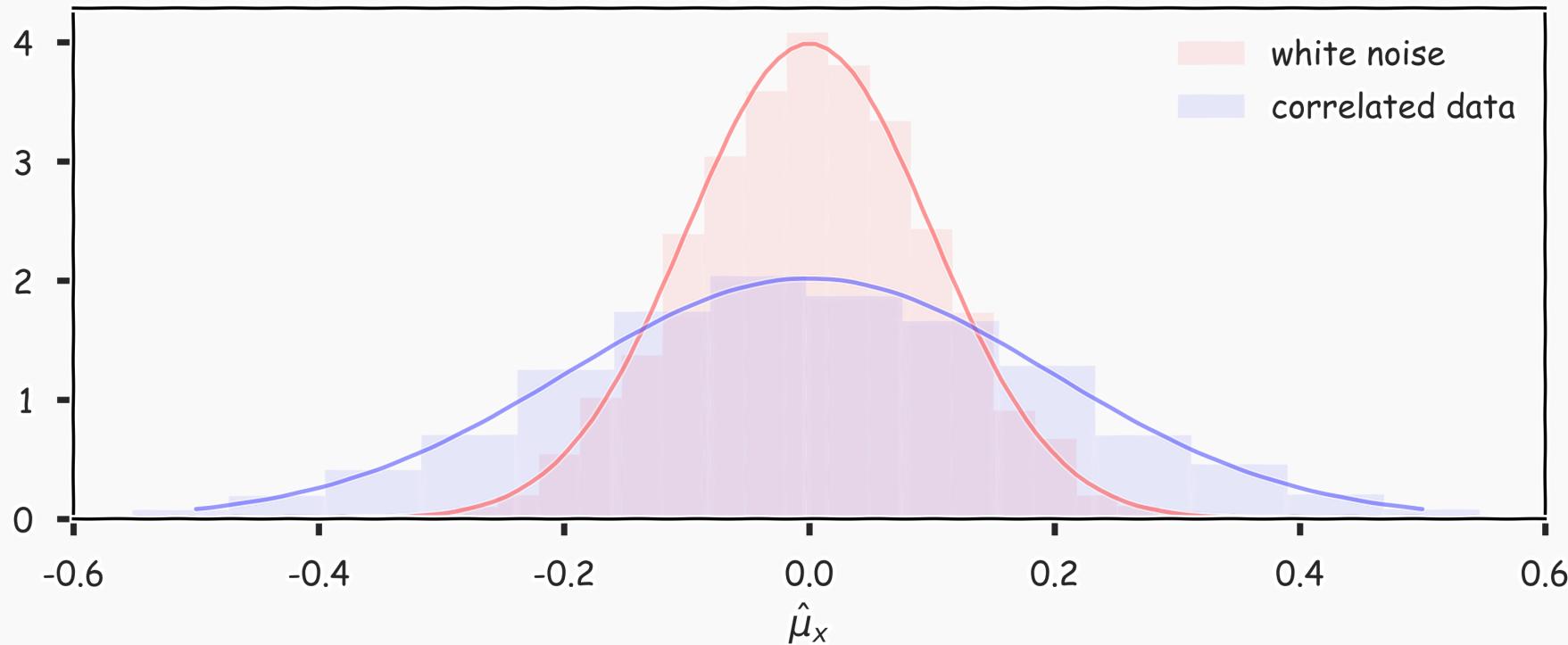


Improving on Bagging

For B number of identically but not independently distributed variables with pairwise correlation ρ and variance σ^2 , the variance of their mean is

$$\text{var}(\hat{\mu}_x) \propto \sigma^2(1 + \rho^2)/B$$

Estimates of means for correlated xs, $\rho = 0.5$, for 100 Xs. Here we show the results for 3000 simulations



Random Forests

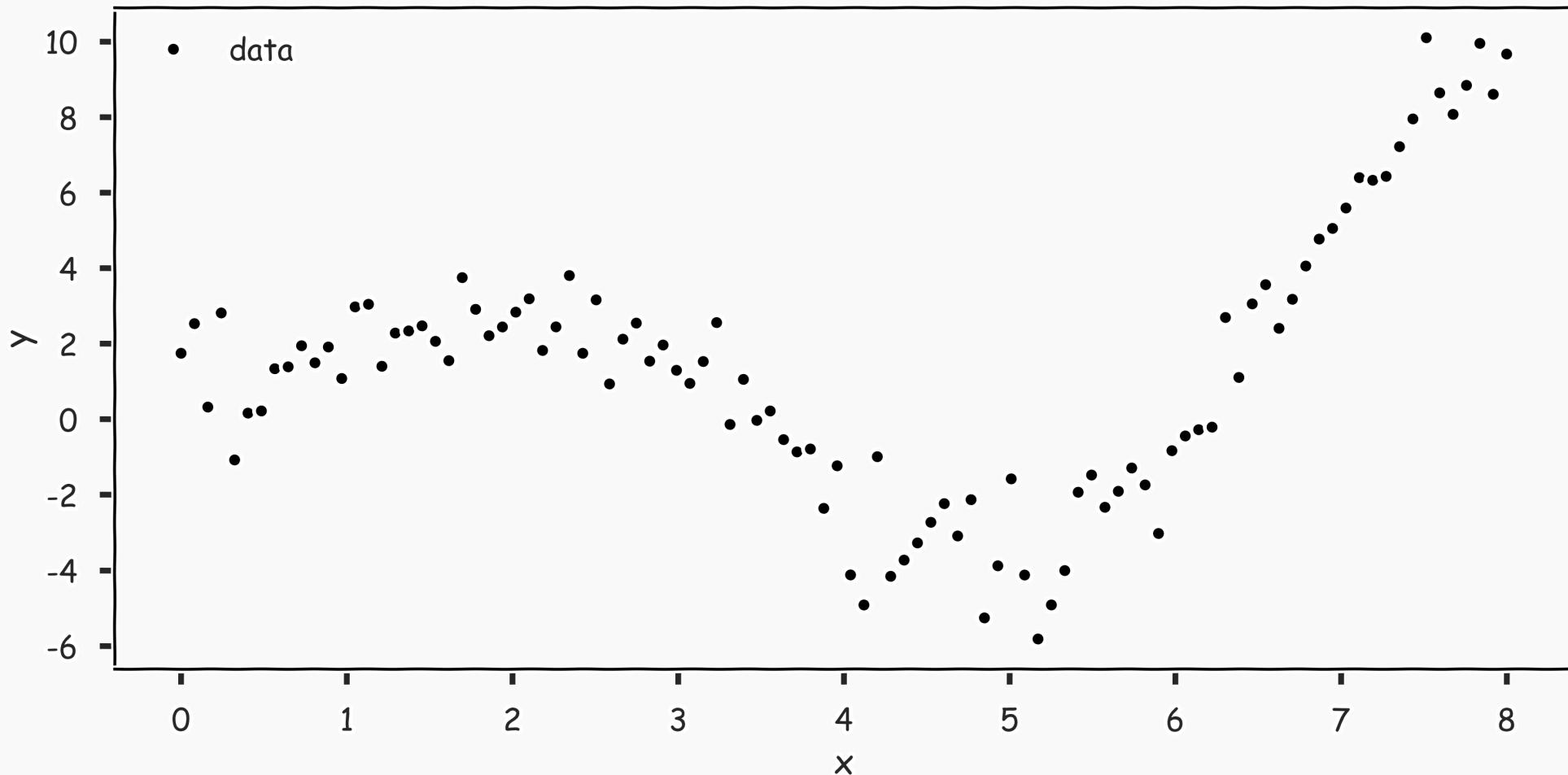
Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

To de-correlate the trees, we:

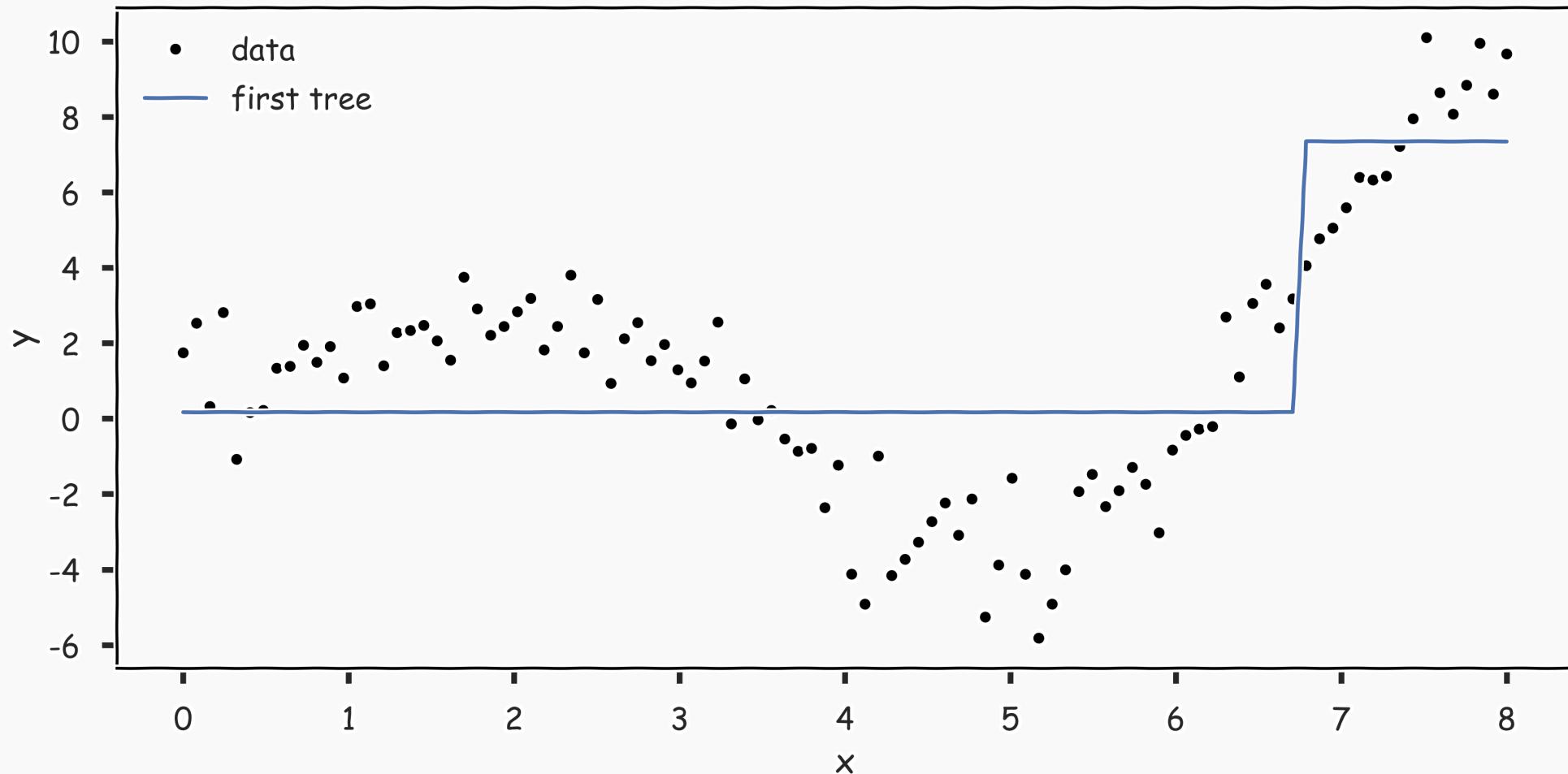
1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)
2. for each tree, at each split, we *randomly* select a set of J' predictors from the full set of predictors.

From amongst the J' predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

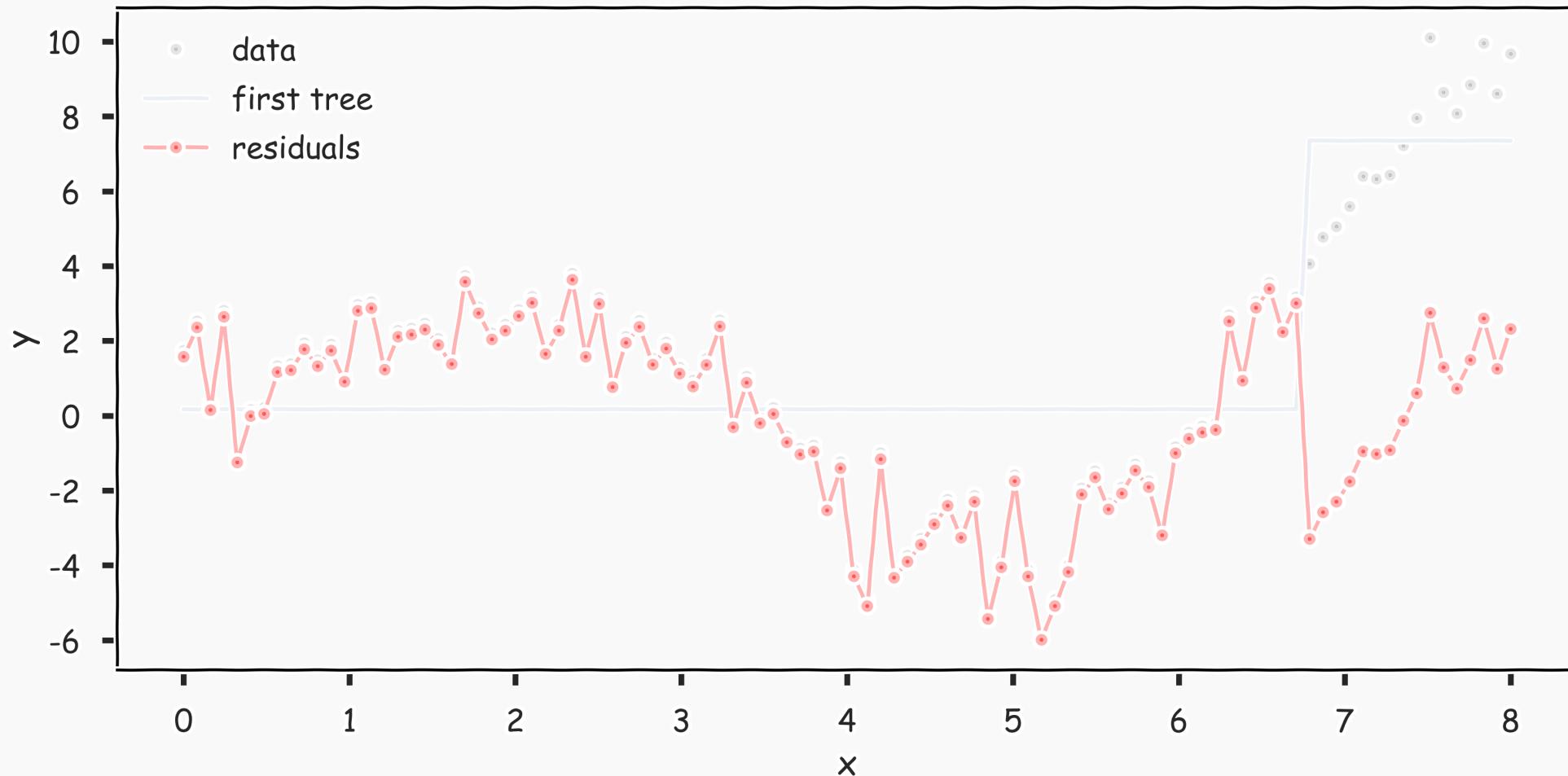
Gradient Boosting: illustration



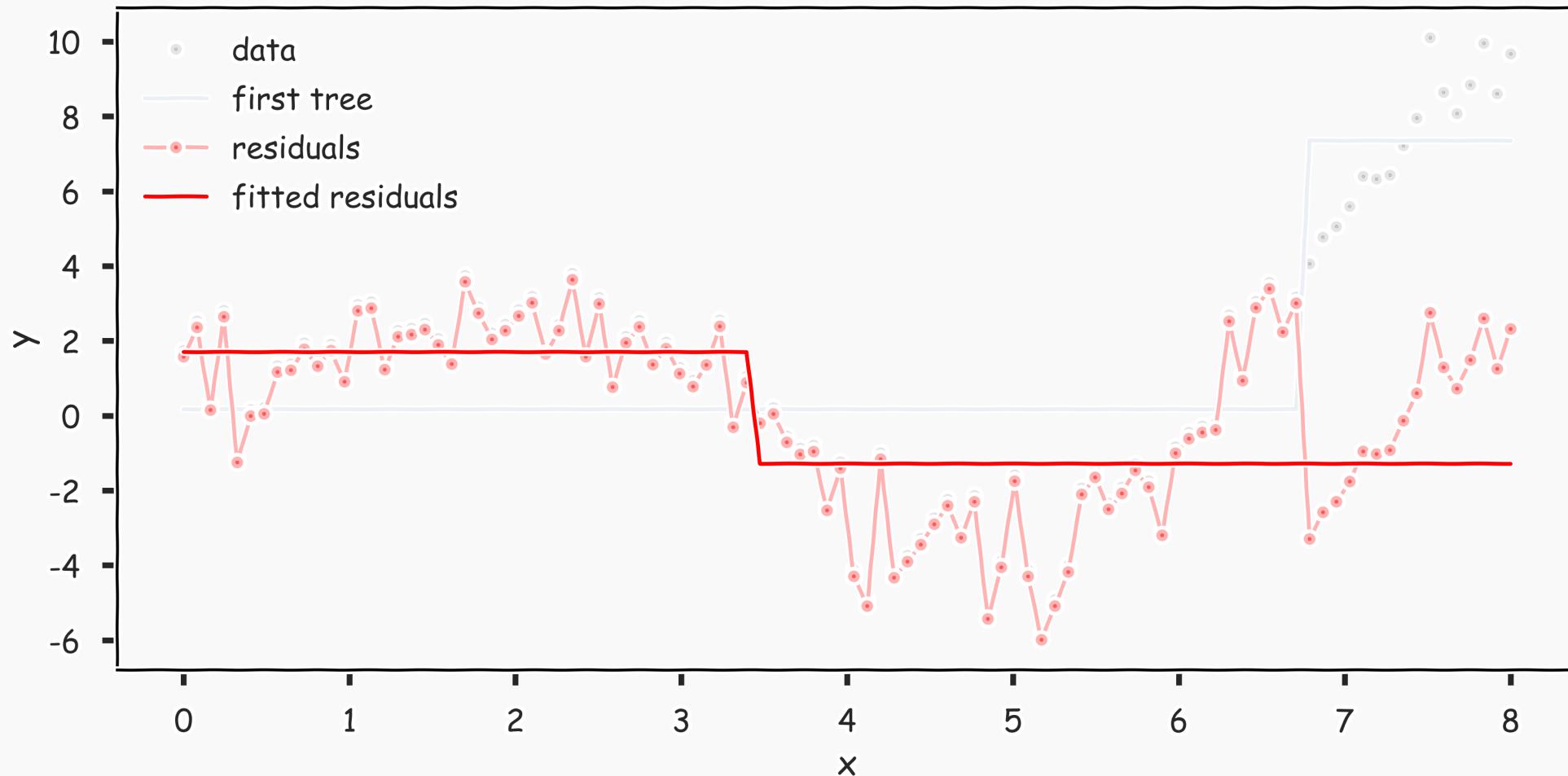
Gradient Boosting: illustration



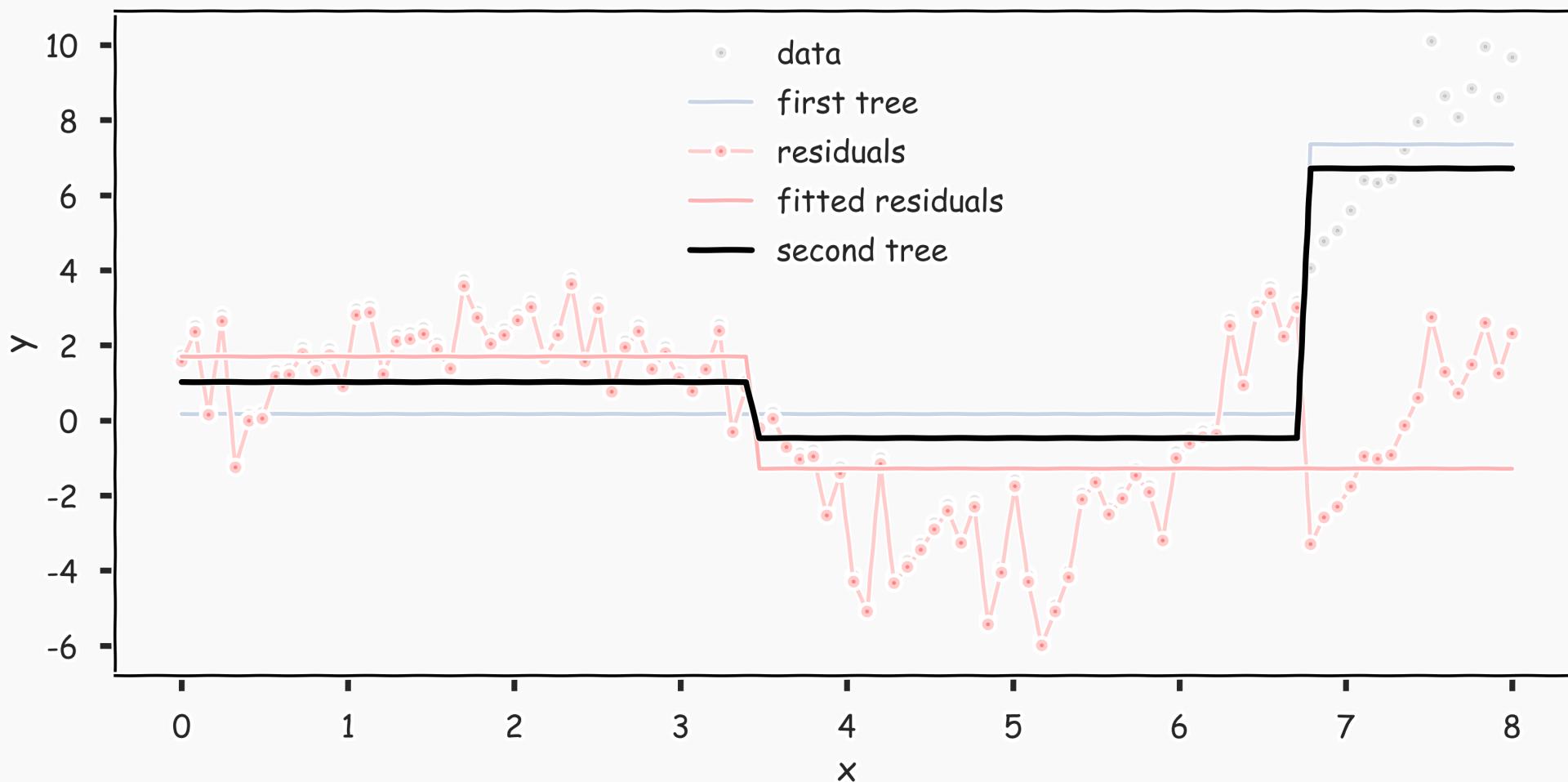
Gradient Boosting: illustration



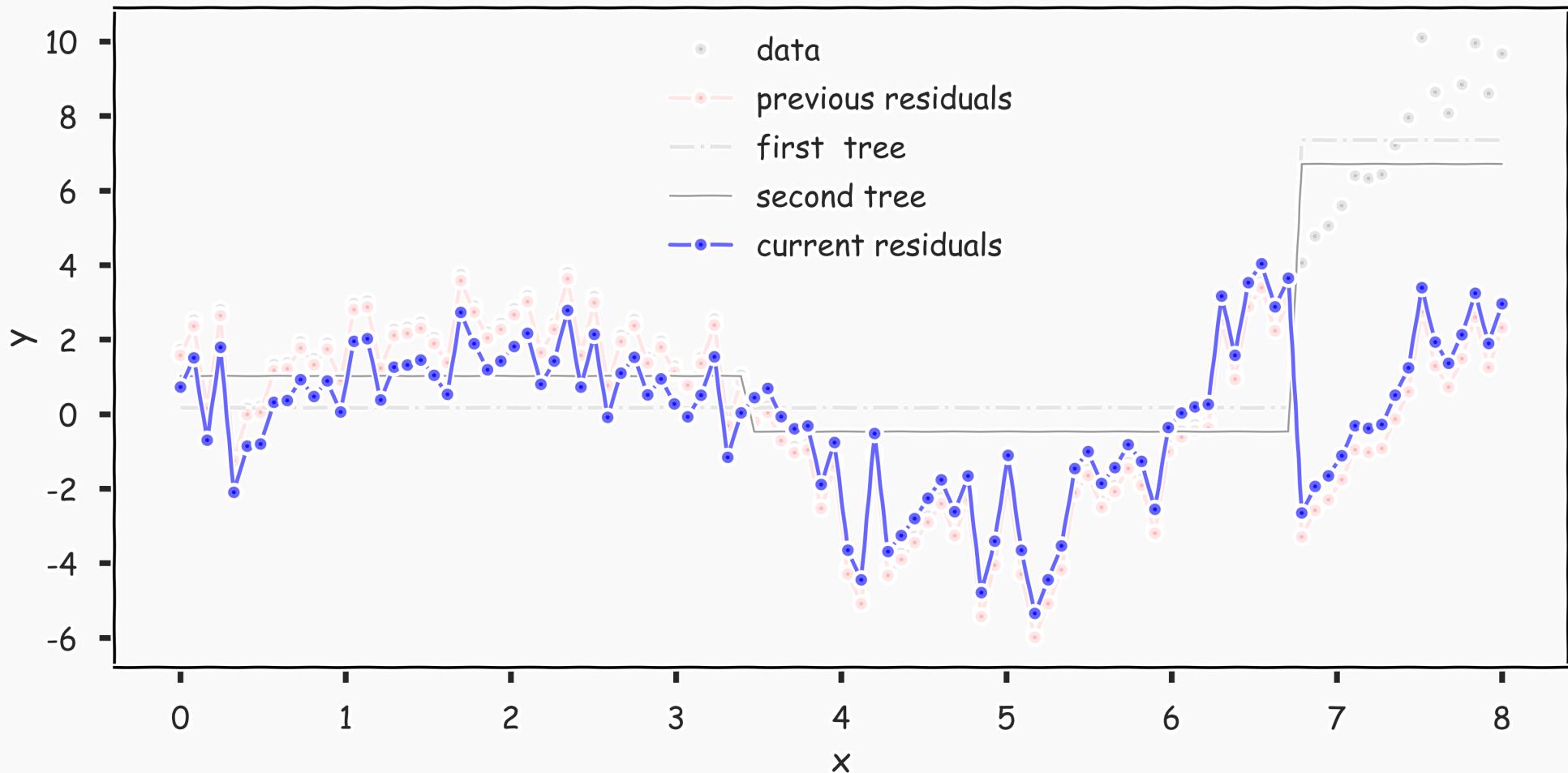
Gradient Boosting: illustration



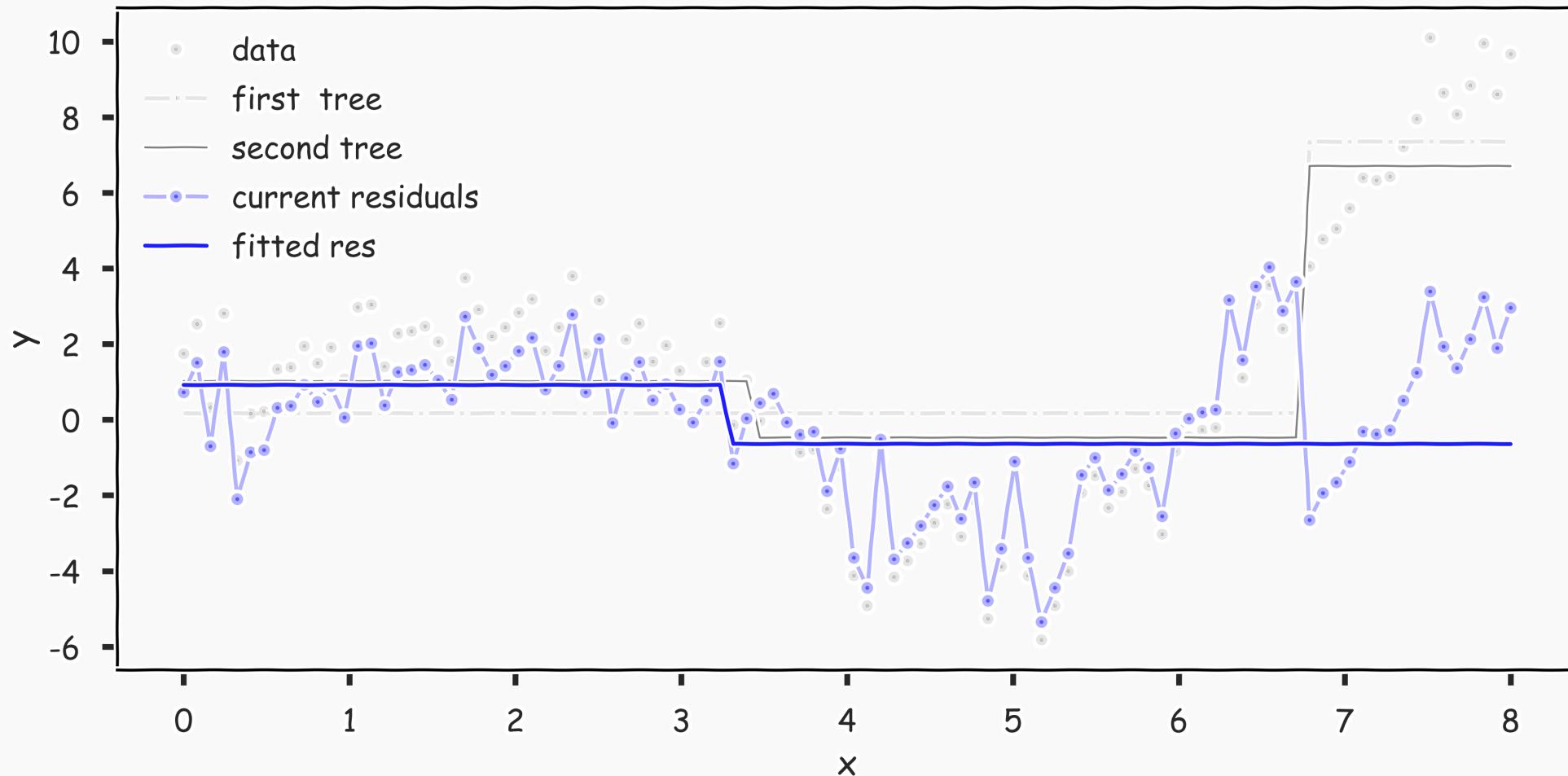
Gradient Boosting: illustration



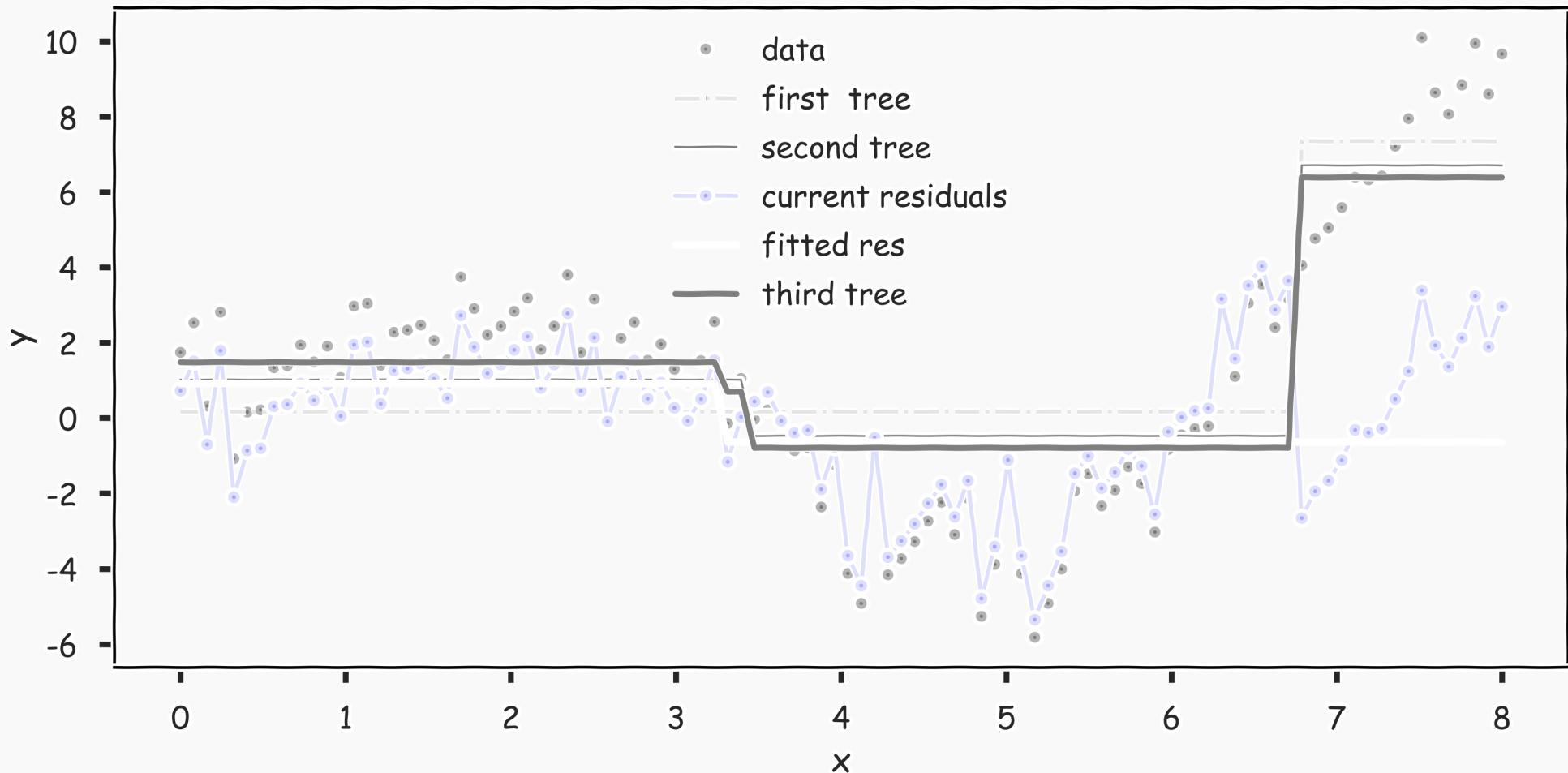
Gradient Boosting: illustration

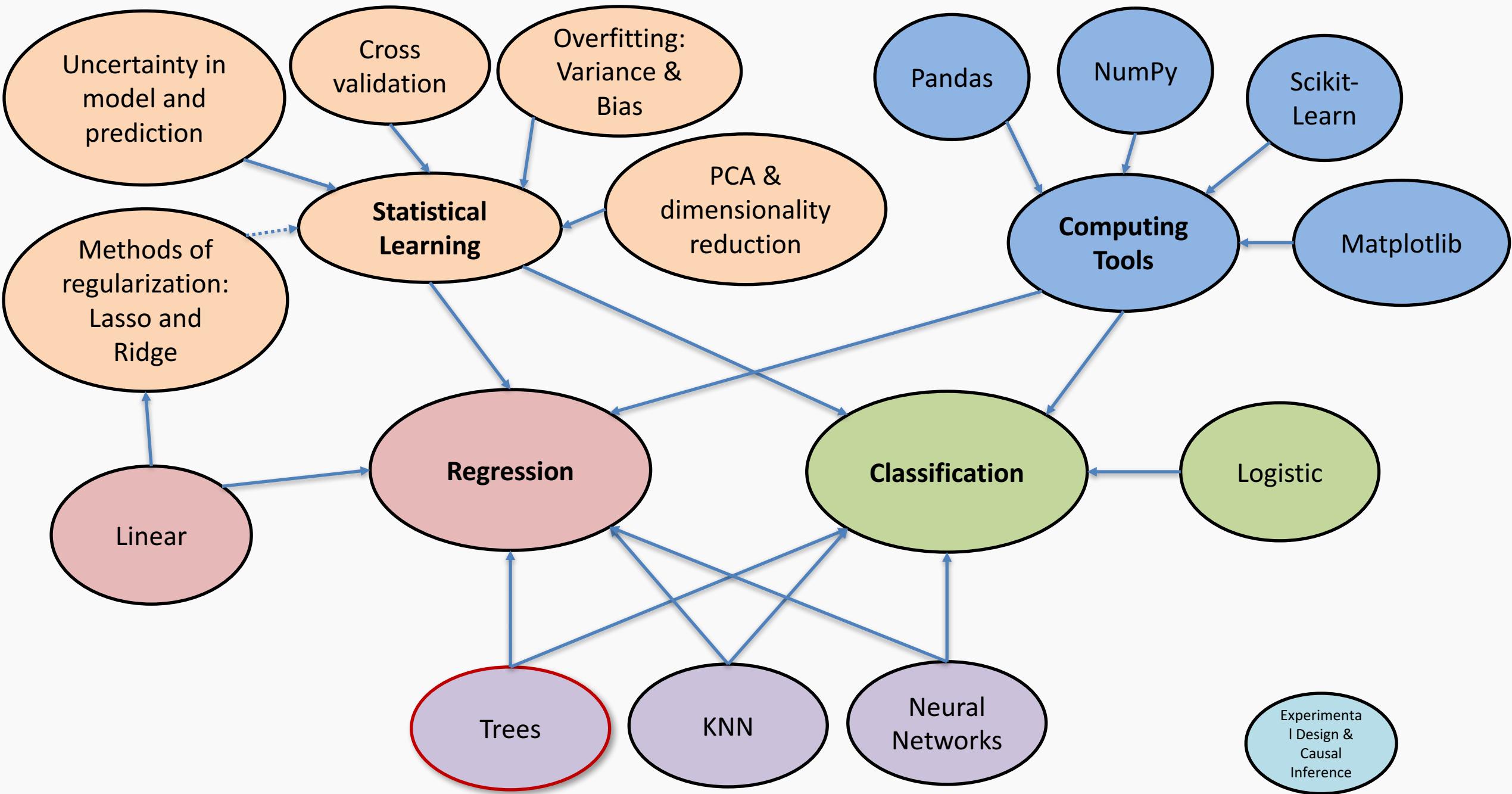


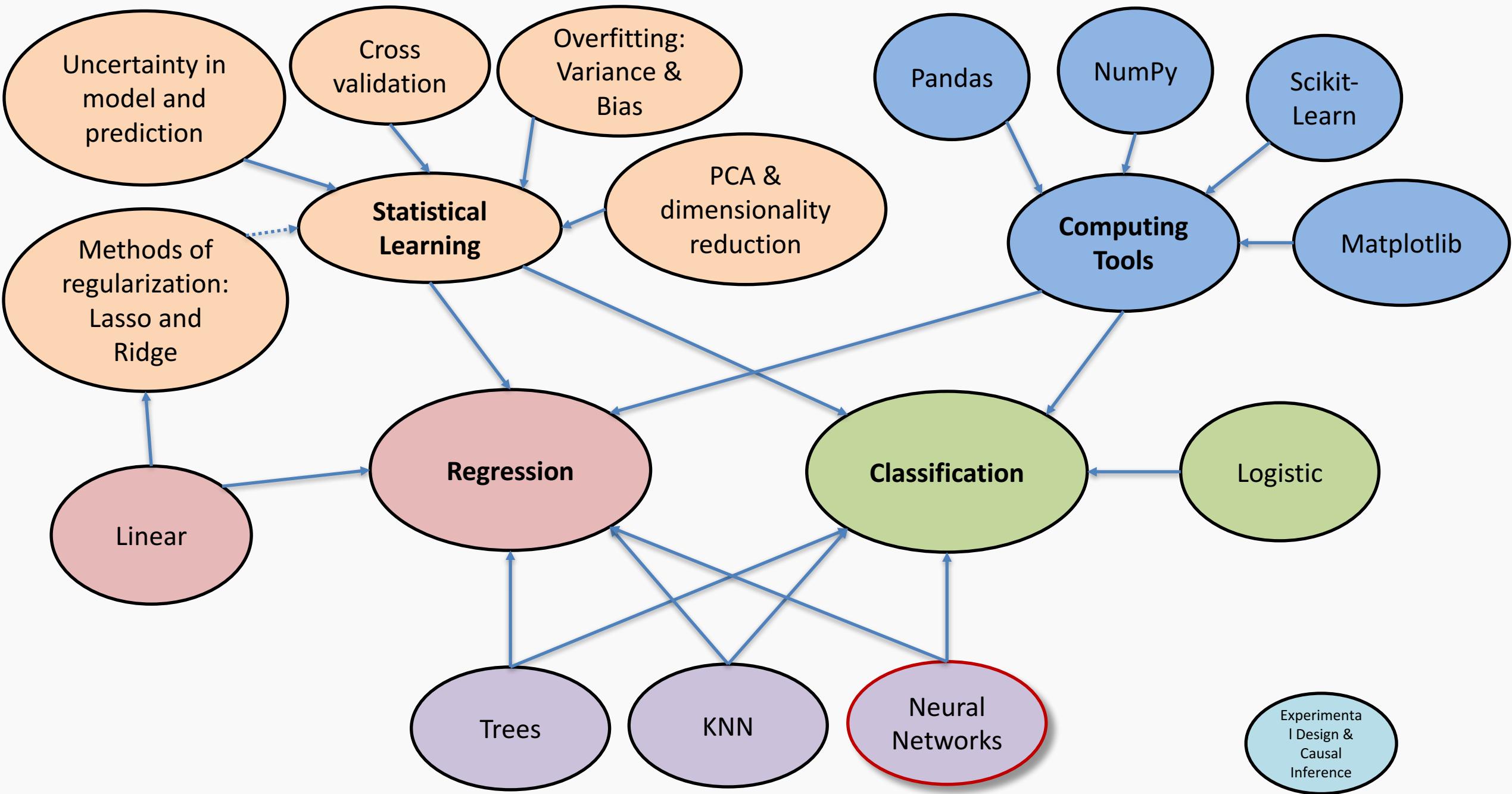
Gradient Boosting: illustration



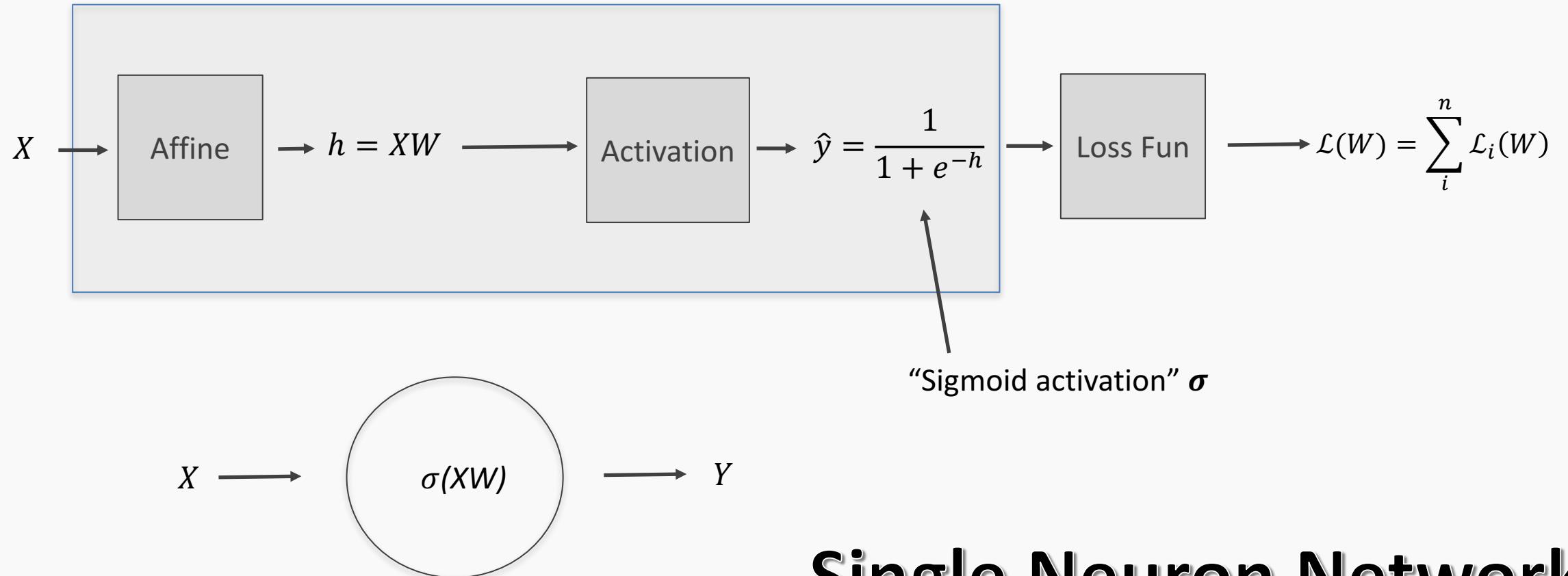
Gradient Boosting: illustration





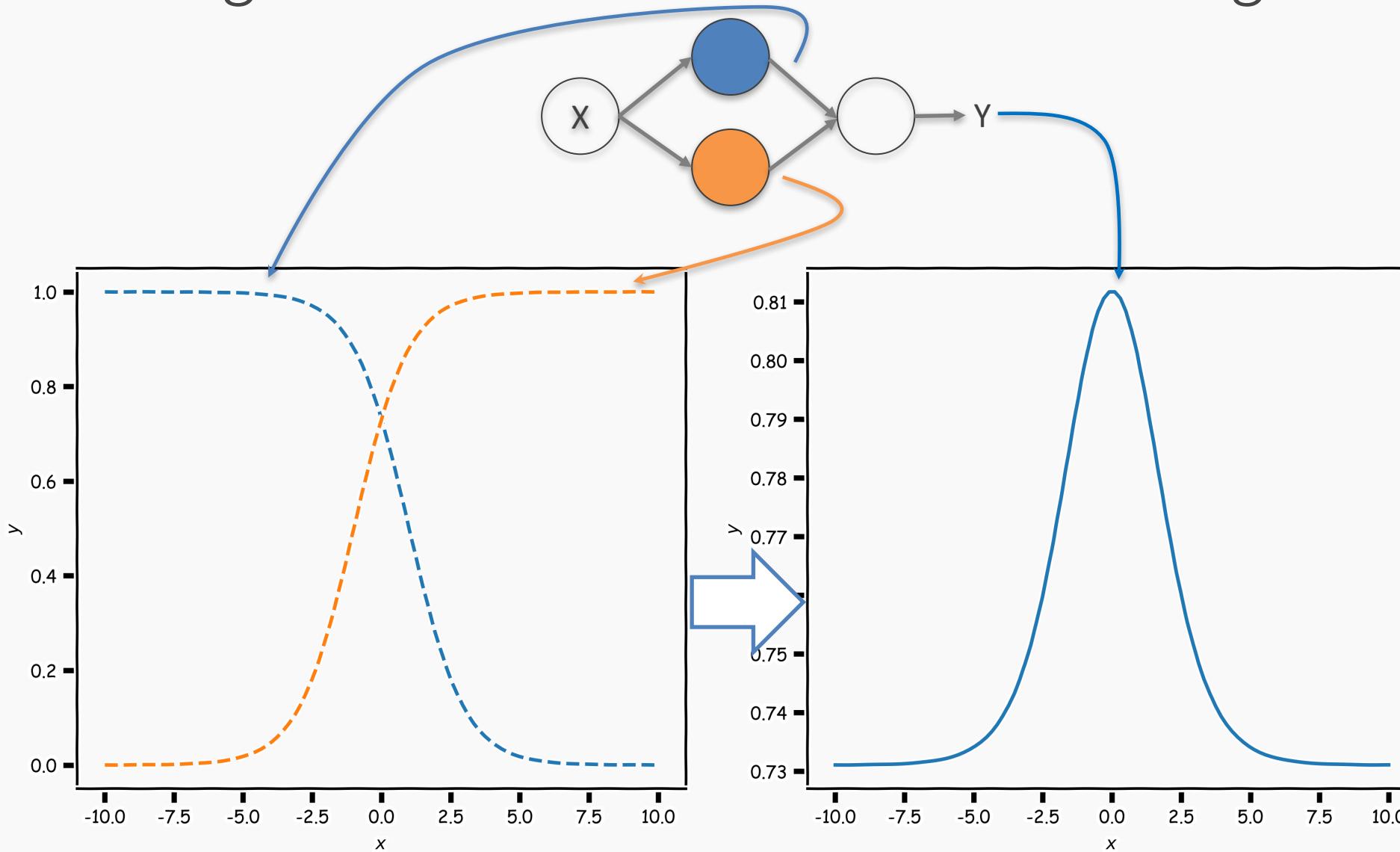


Build our first ANN

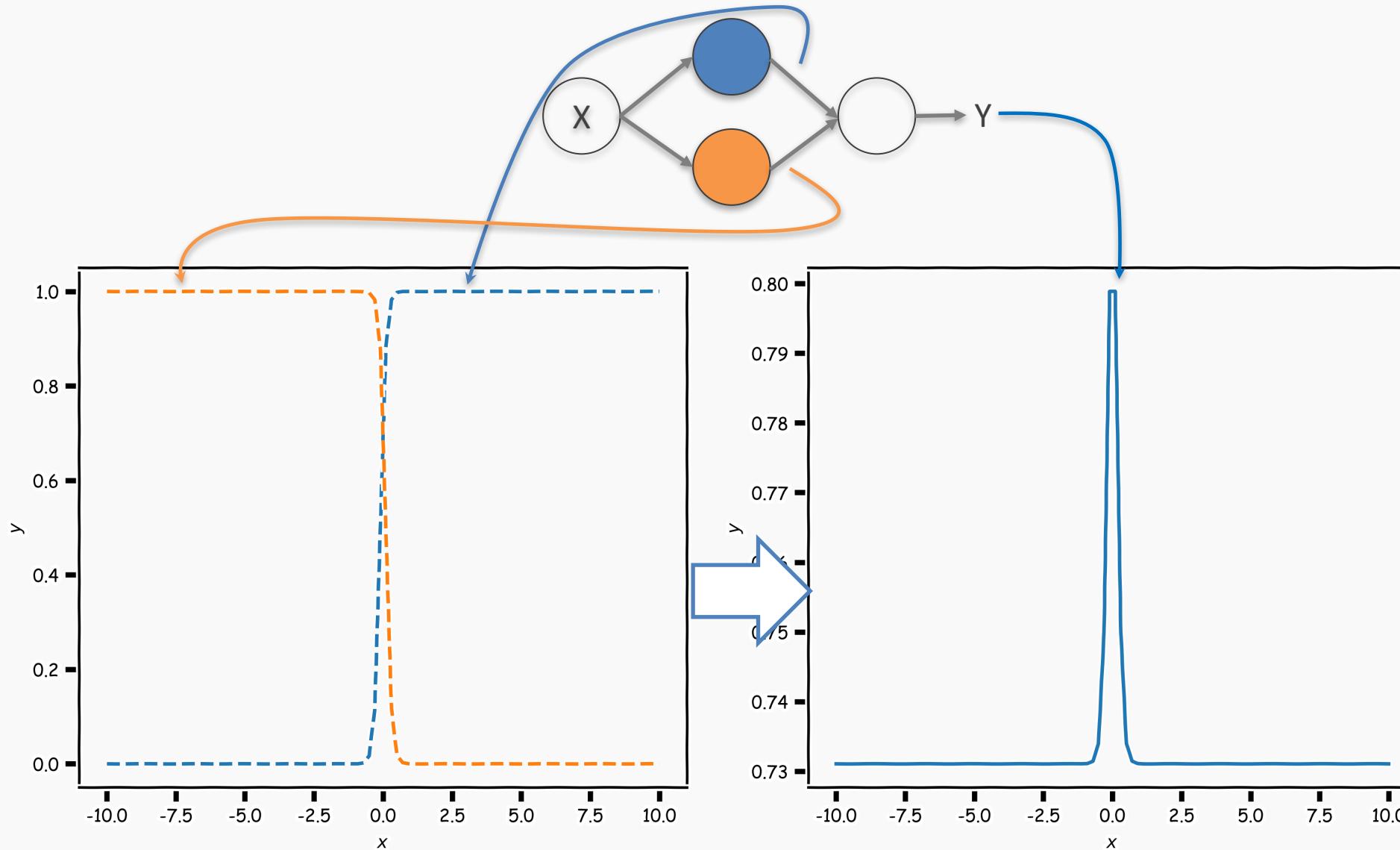


Single Neuron Network
Very similar to Perceptron

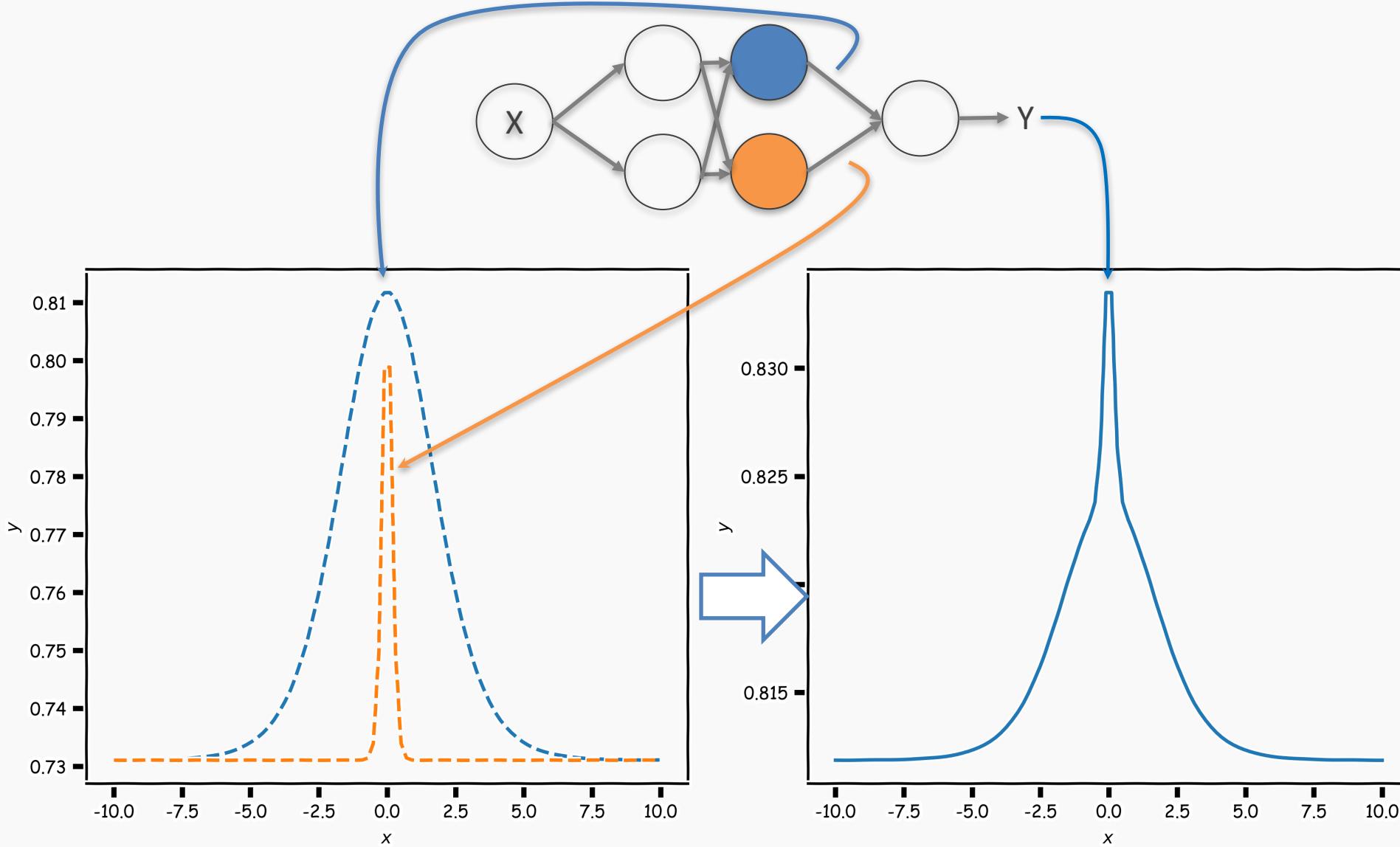
Combining neurons allows us to model interesting functions



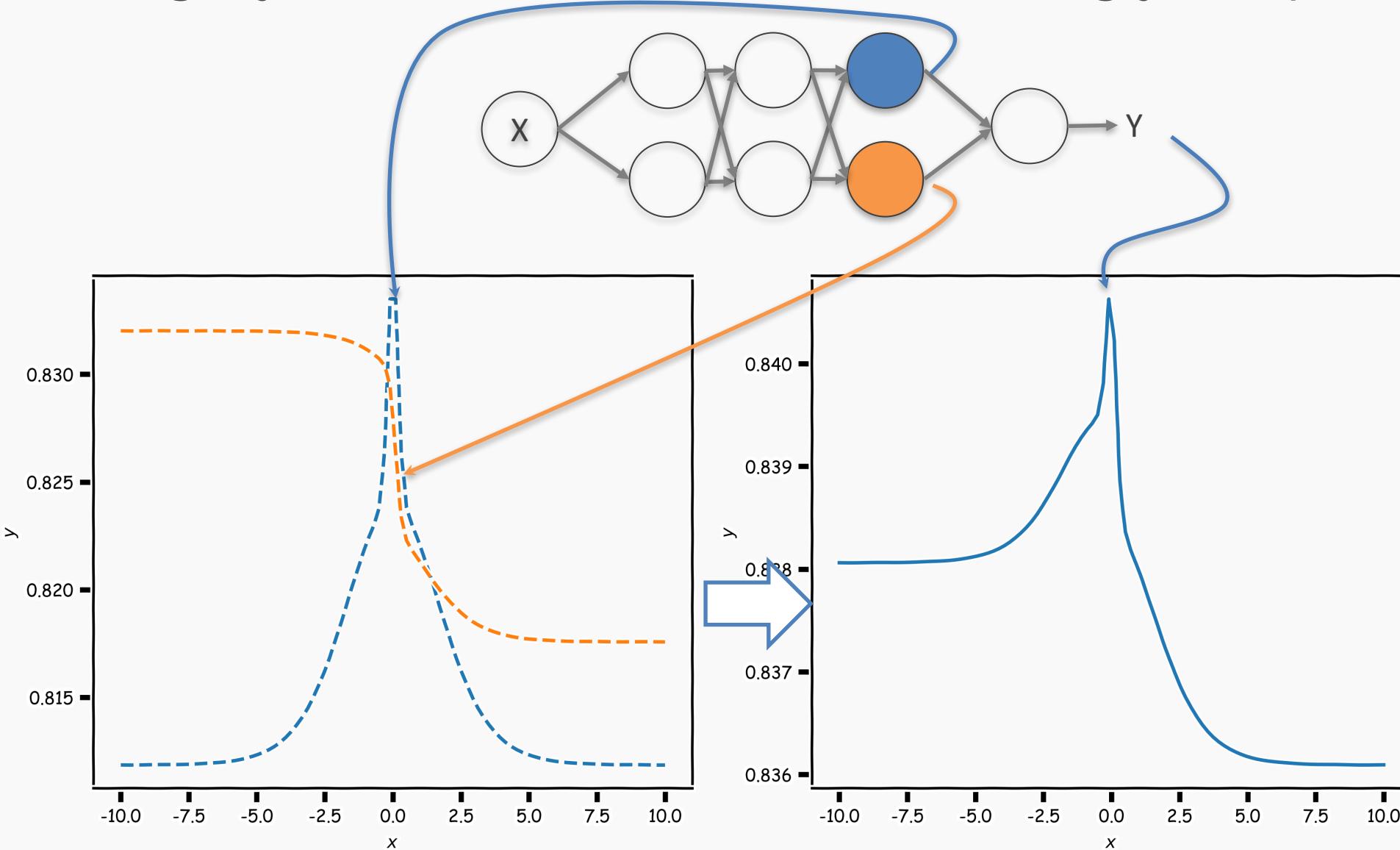
Different weights change the shape and position



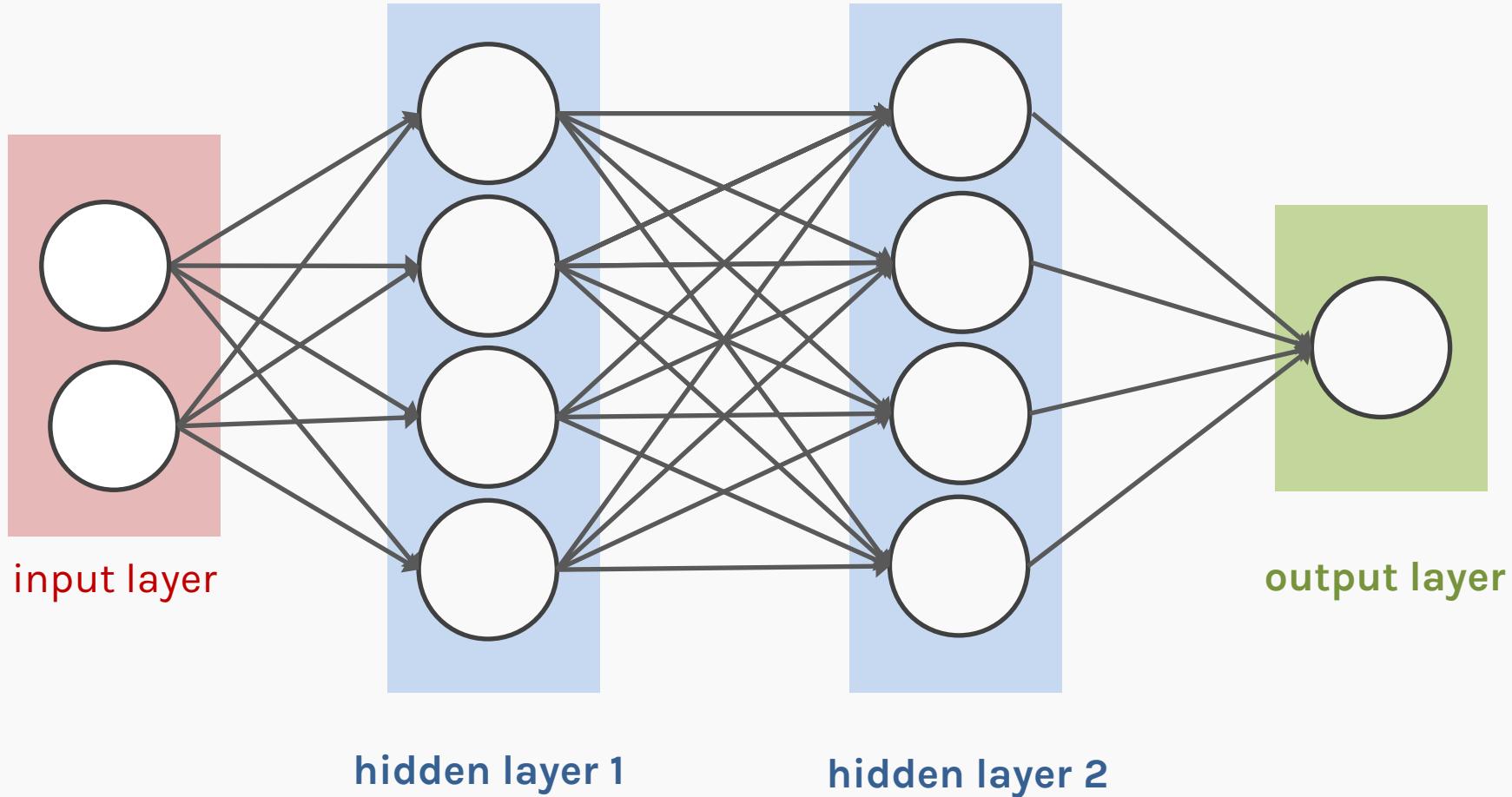
Neural networks can model any reasonable function



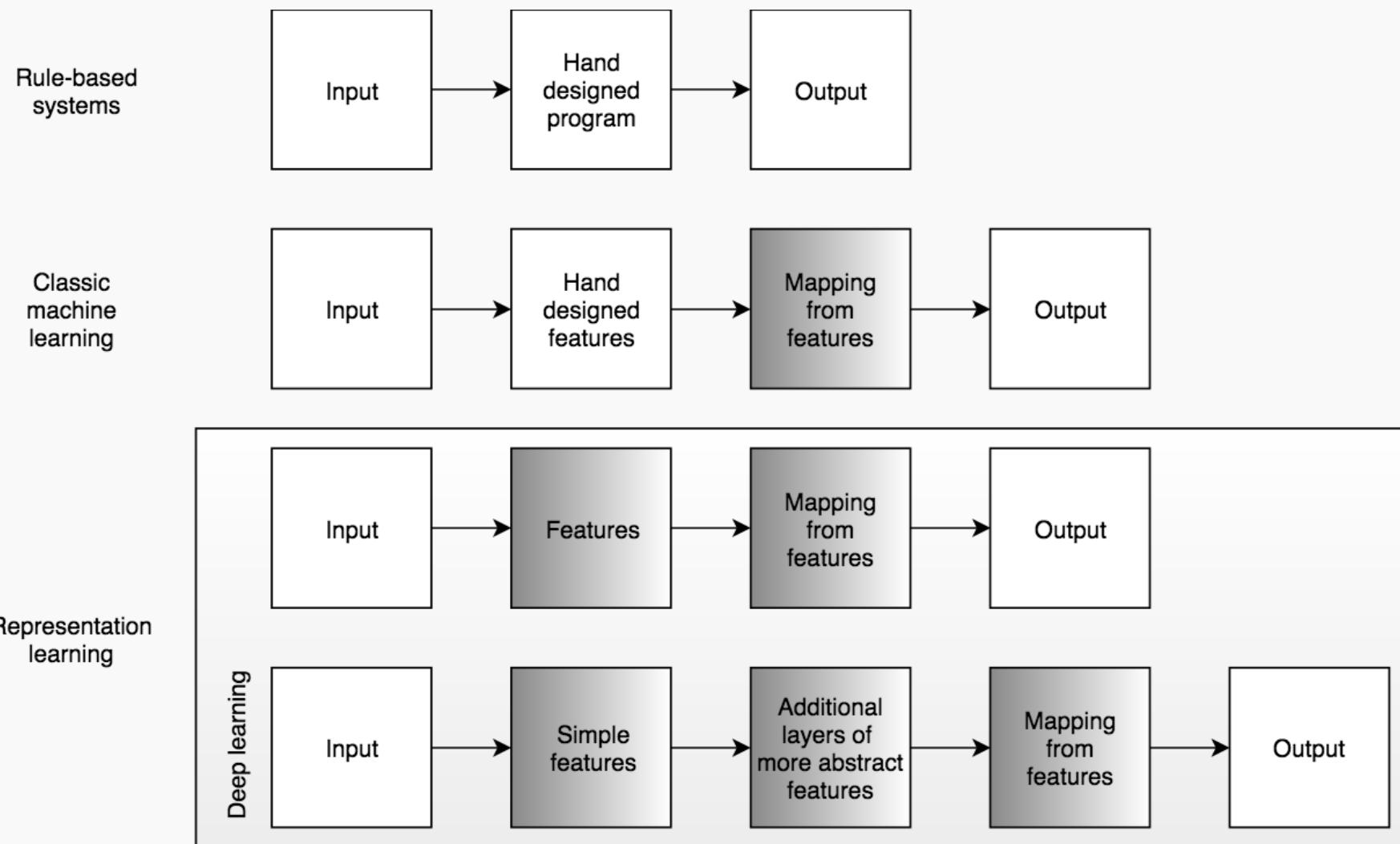
Adding layers allows us to model increasingly complex functions



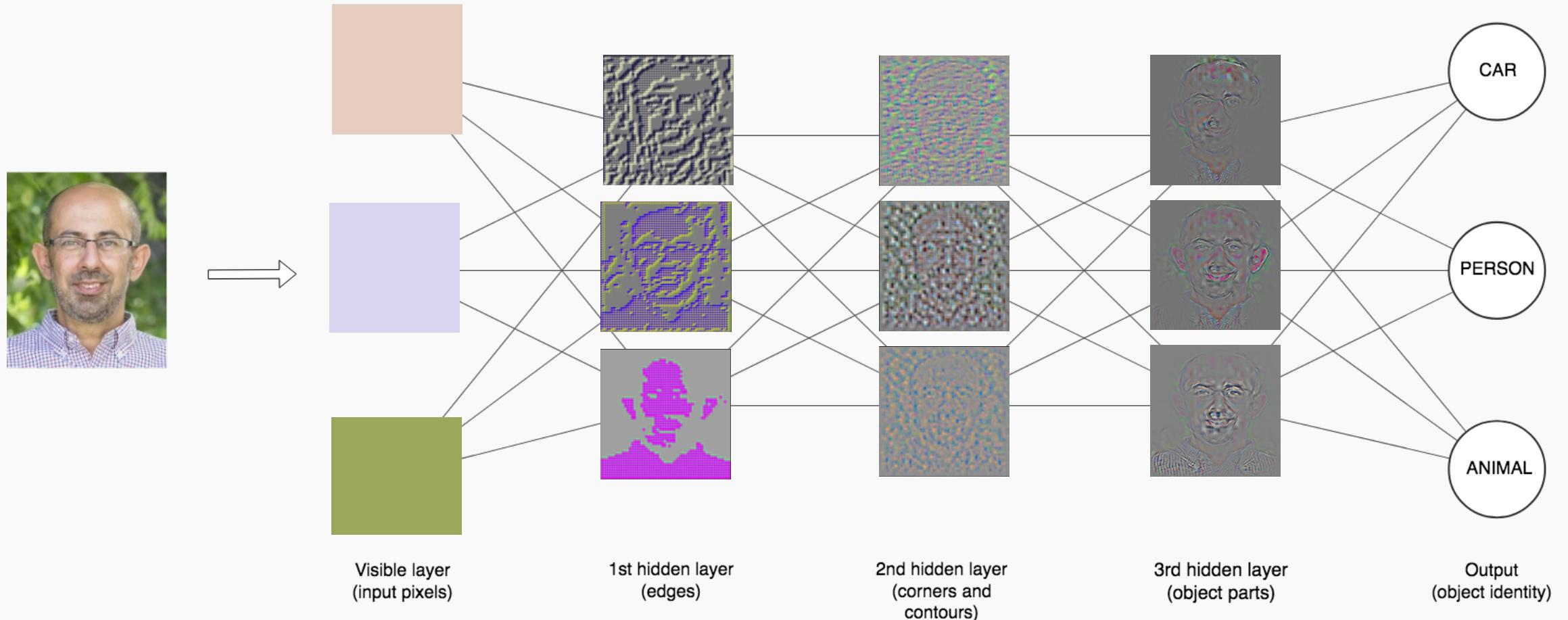
Anatomy of artificial neural network (ANN)



Learning Multiple Components

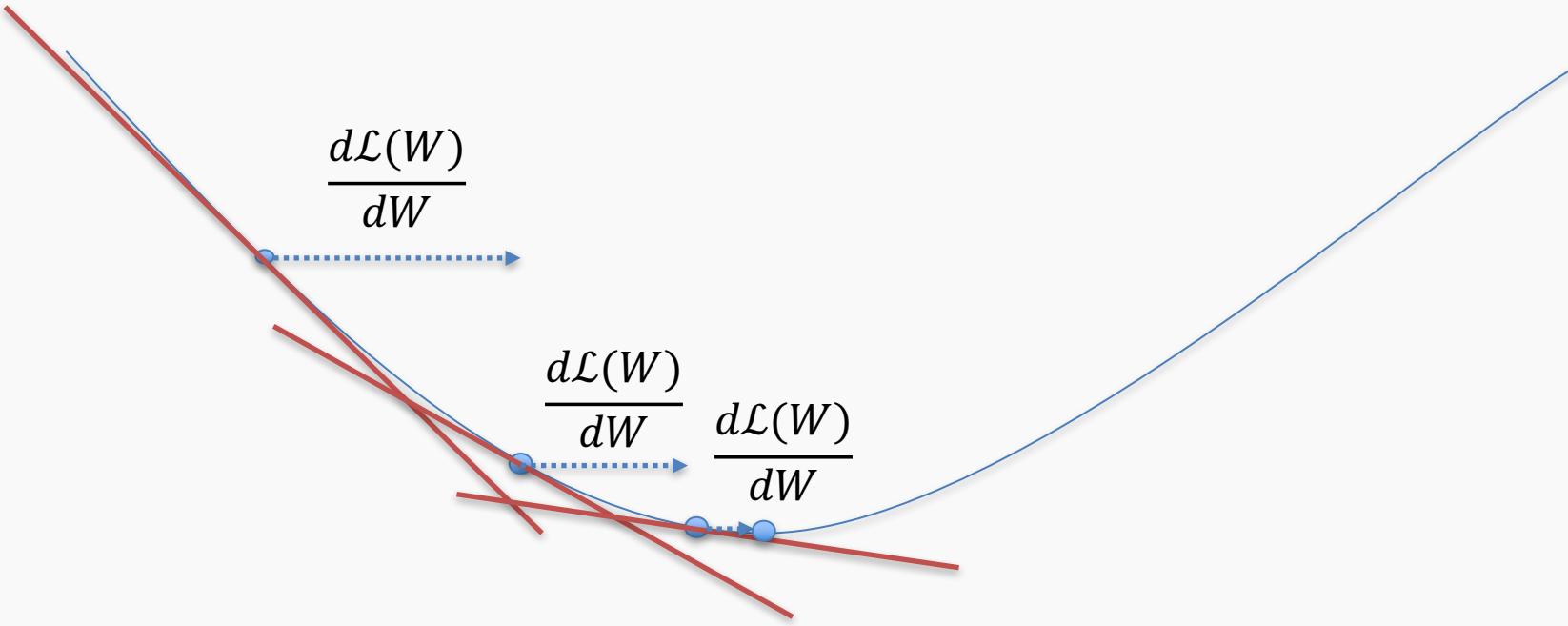


Depth = Repeated Compositions



Gradient Descent (cont.)

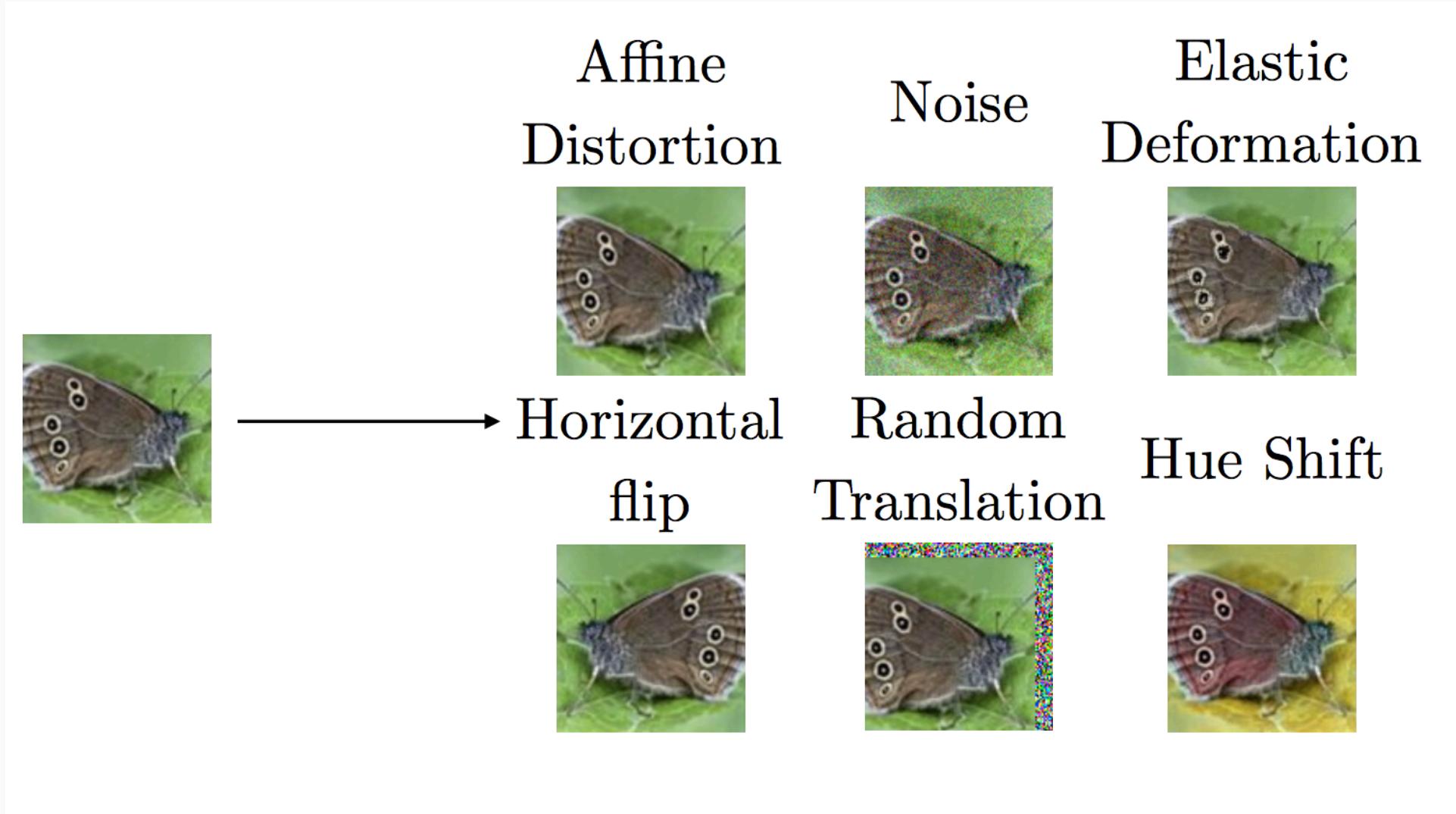
If the step is proportional to the slope then you avoid overshooting the minimum. How?



Data Augmentation

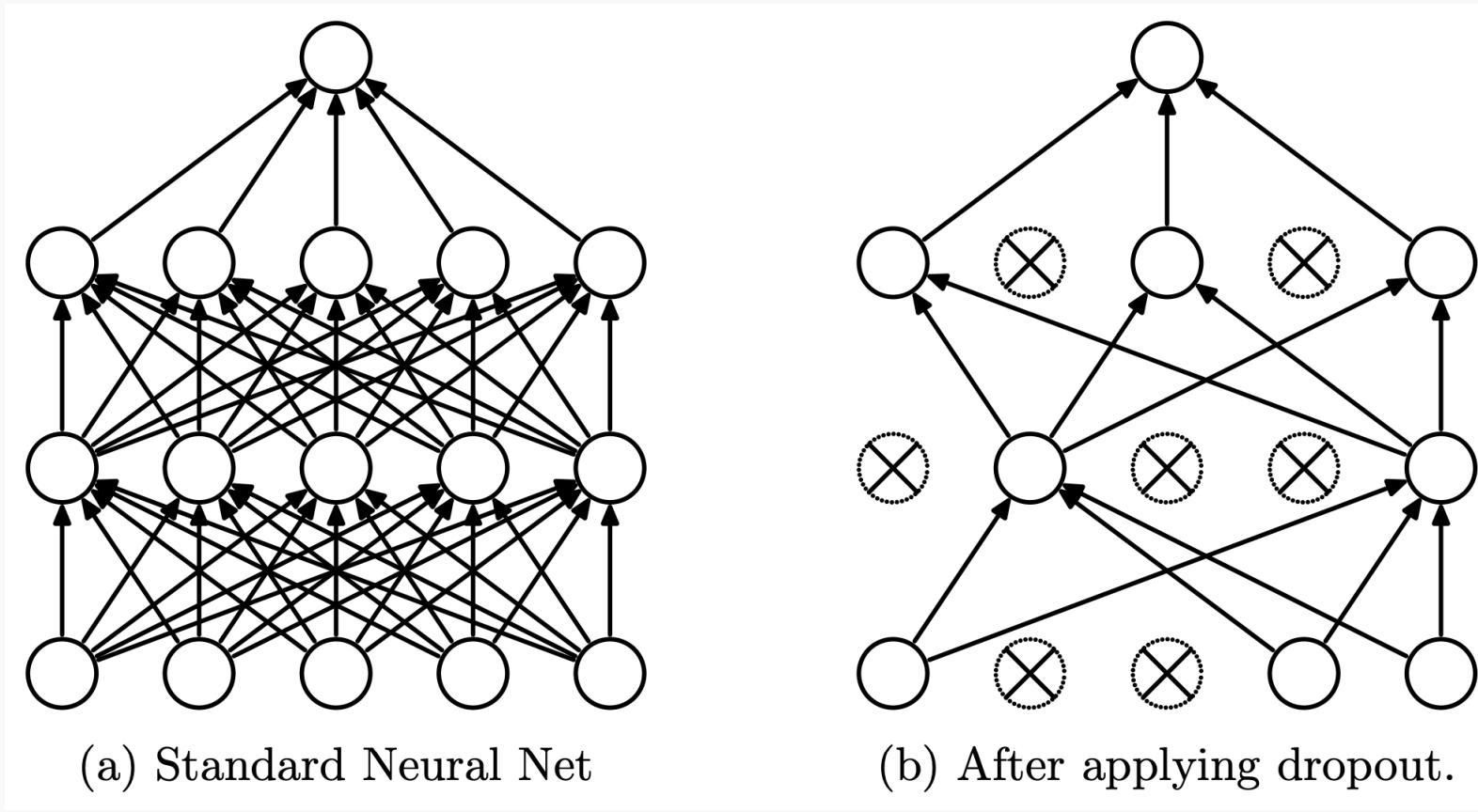


Data Augmentation



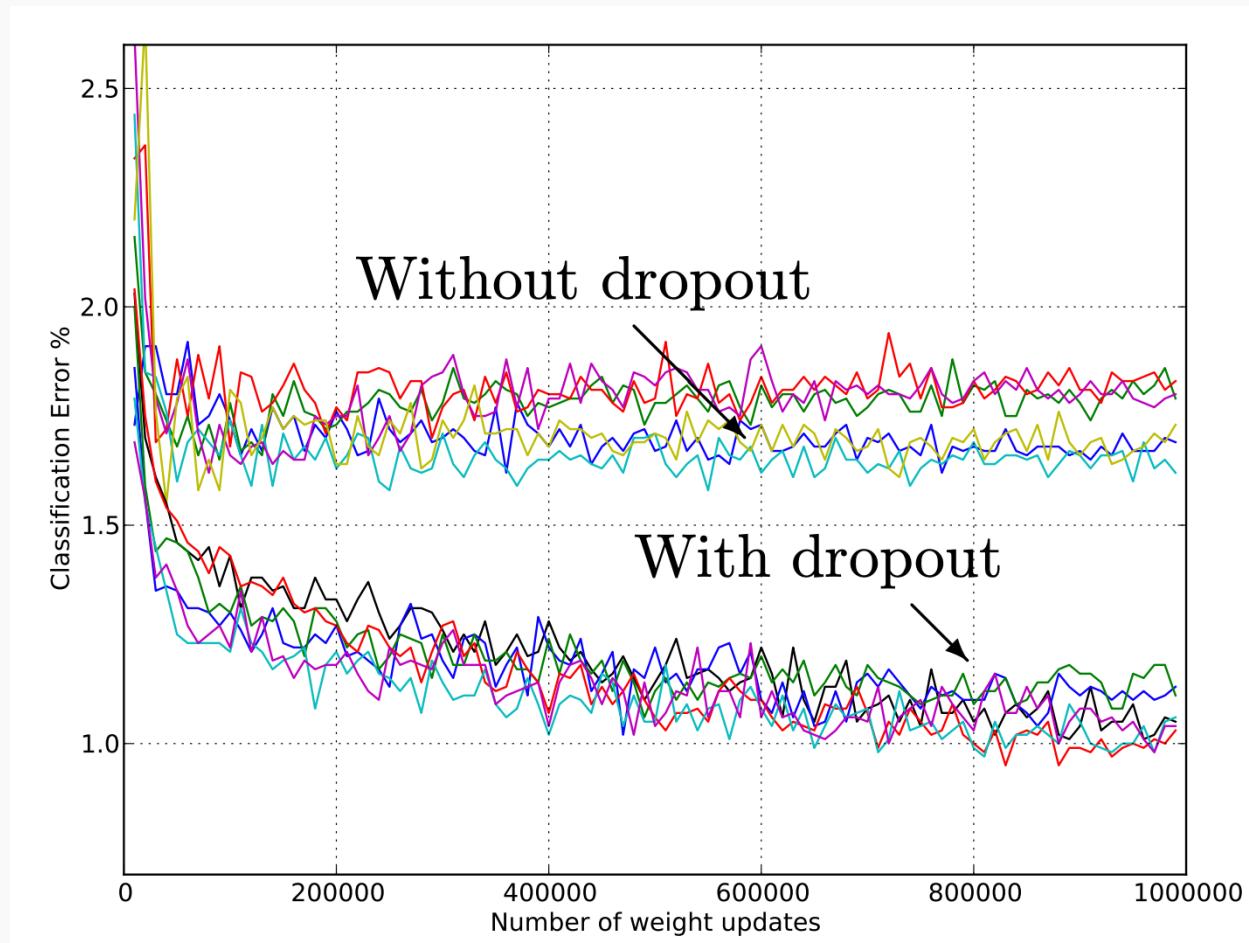
Dropout

- Randomly set some neurons and their connections to zero (i.e. “dropped”)
- Prevent overfitting by reducing co-adaptation of neurons
- Like training many random sub-networks

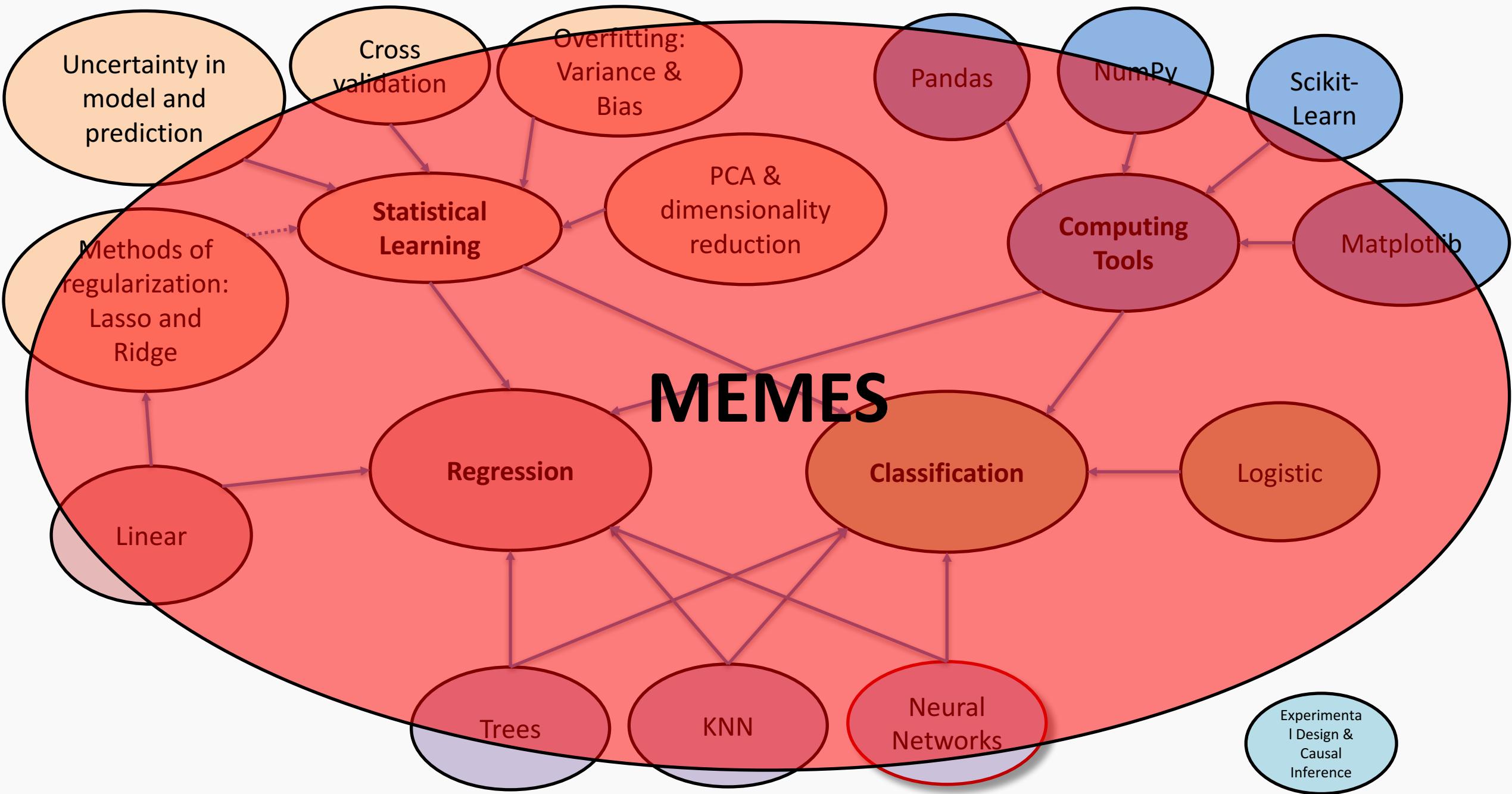


Dropout

- Widely used and highly effective
- Proposed as an alternative to ensembling, which is too expensive for neural nets



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.



Summary from last lecture

When you're a kNN with $k = 2$



Confidence intervals for the predictors estimates (cont)

So if we just have one set of measurements of $\{X, Y\}$, our estimates of $\hat{\beta}_0$ and $\hat{\beta}_1$ are just for this particular realization.





You're Gonna Have a Bad Time...





Quiz time: Lecture 8



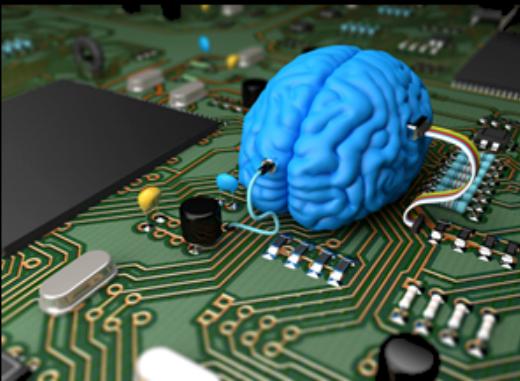


PPppBoost

Deep Learning



What society thinks I do



What my friends think I do



What other computer
scientists think I do



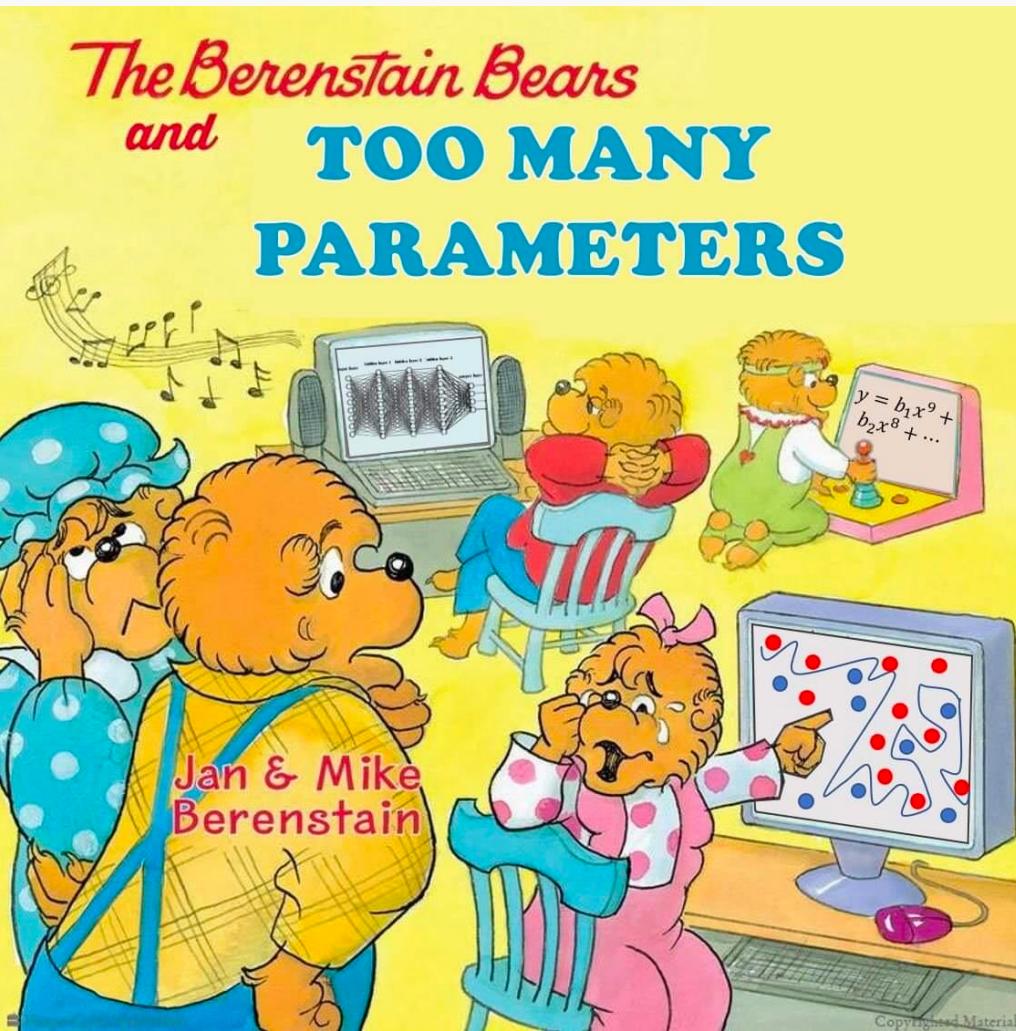
What mathematicians think I do



What I think I do

```
In [1]:  
import keras  
Using TensorFlow backend.
```

What I actually do



Better than AB...



Scalar



Vector



Matrix

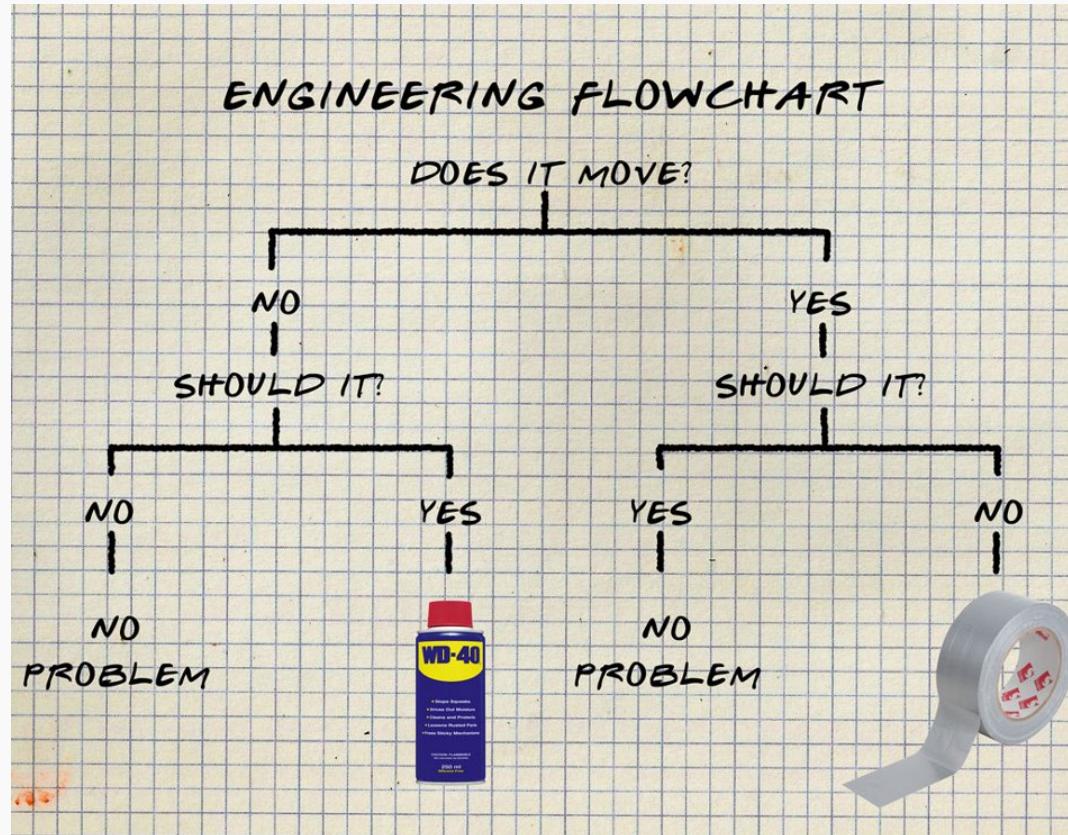


Tensor

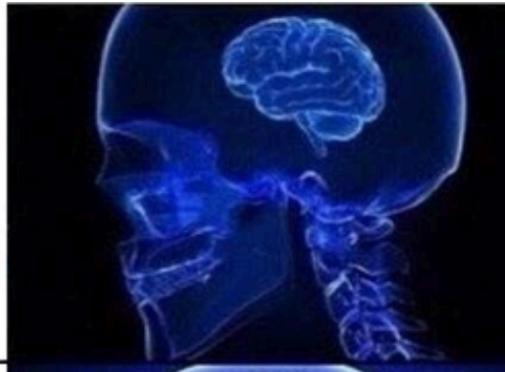


Interpretable Models

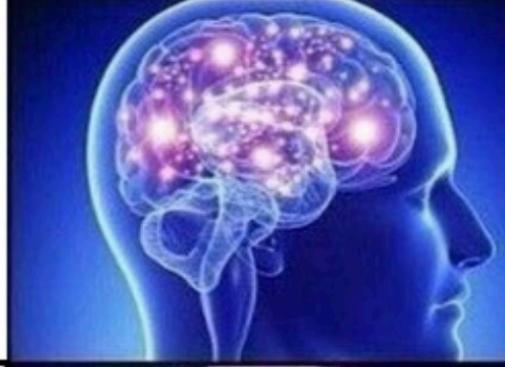
People in every walk of life have long been using interpretable models for differentiating between classes of objects and phenomena:



**INCLUDING
DROPOUT LAYERS**



**SCHEDULING
THE
LEARNING RATE**



**EXPERIMENTING
WITH
ACTIVATION FUNCTIONS**



**OPTIMIZING
THE RANDOM SEED**

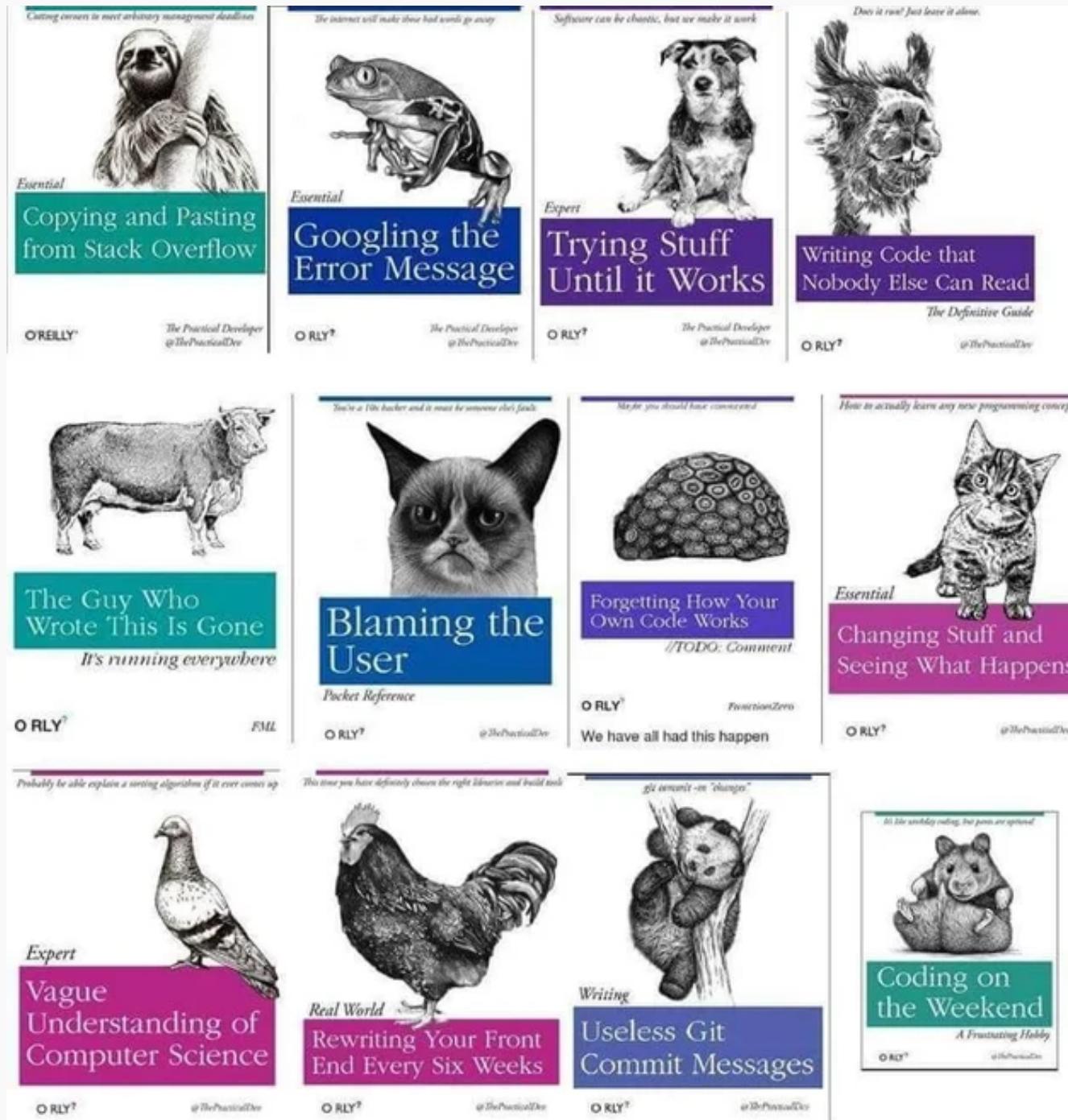


BOOTSTRAPPING

I DONT THINK IT MEANS WHAT YOU
THINK IT MEANS

memegenerator.net





ANNOUNCEMENTS

- Homework 7 individual



HW1



HW4



HW7

Without OH or ED forum

ANNOUNCEMENTS

- After CS109B, STAT 139 ...



RECURRENT NEURAL NETWORKS

SO HOT RIGHT NOW

imgflip.com



You will be
UNAWARE
OF WHAT I'M SAYING
for **4 OUT OF** *the next* **8 MINUTES**

Zeenat Potia



FIND ME?



Thank You!

