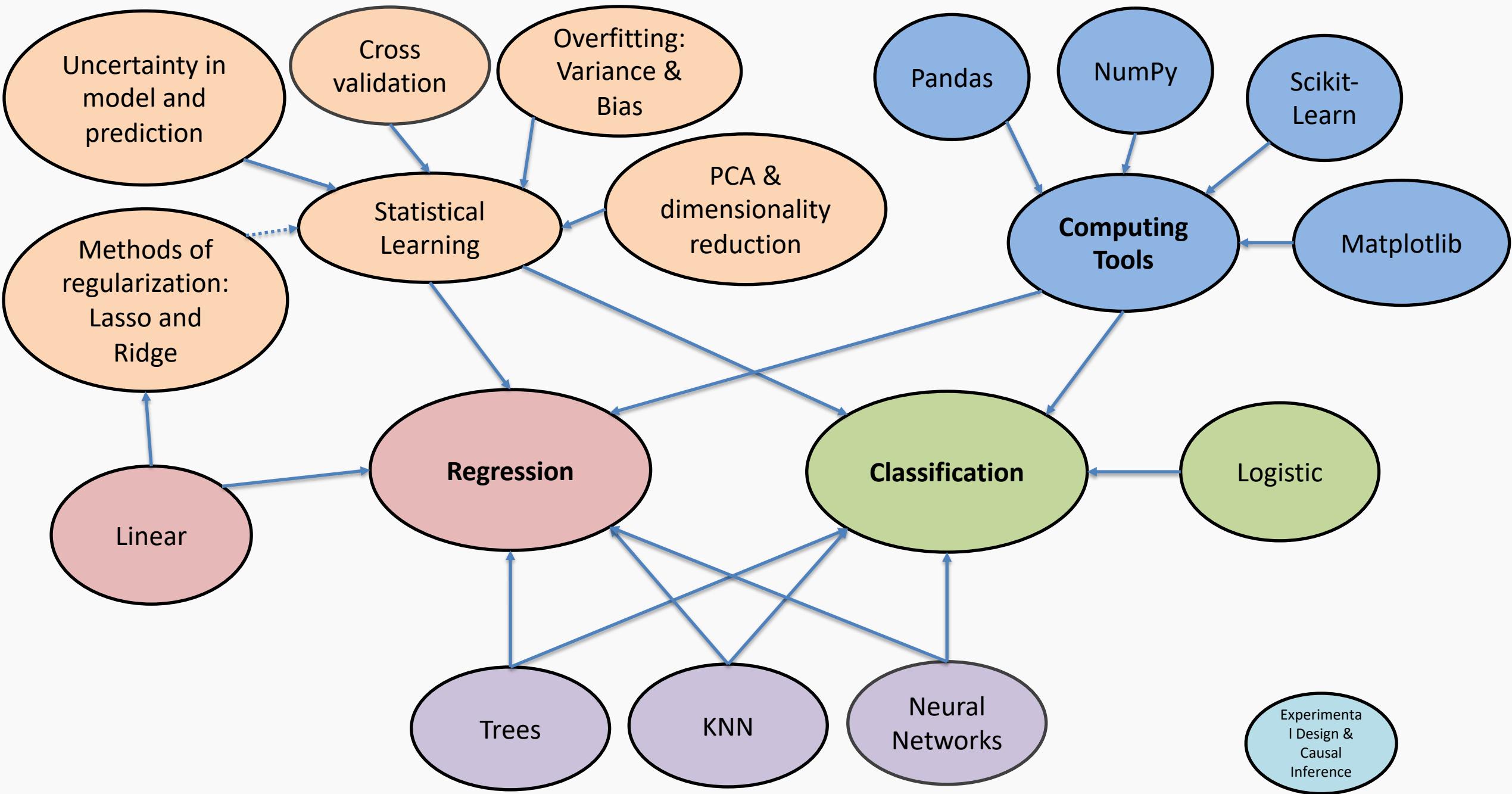
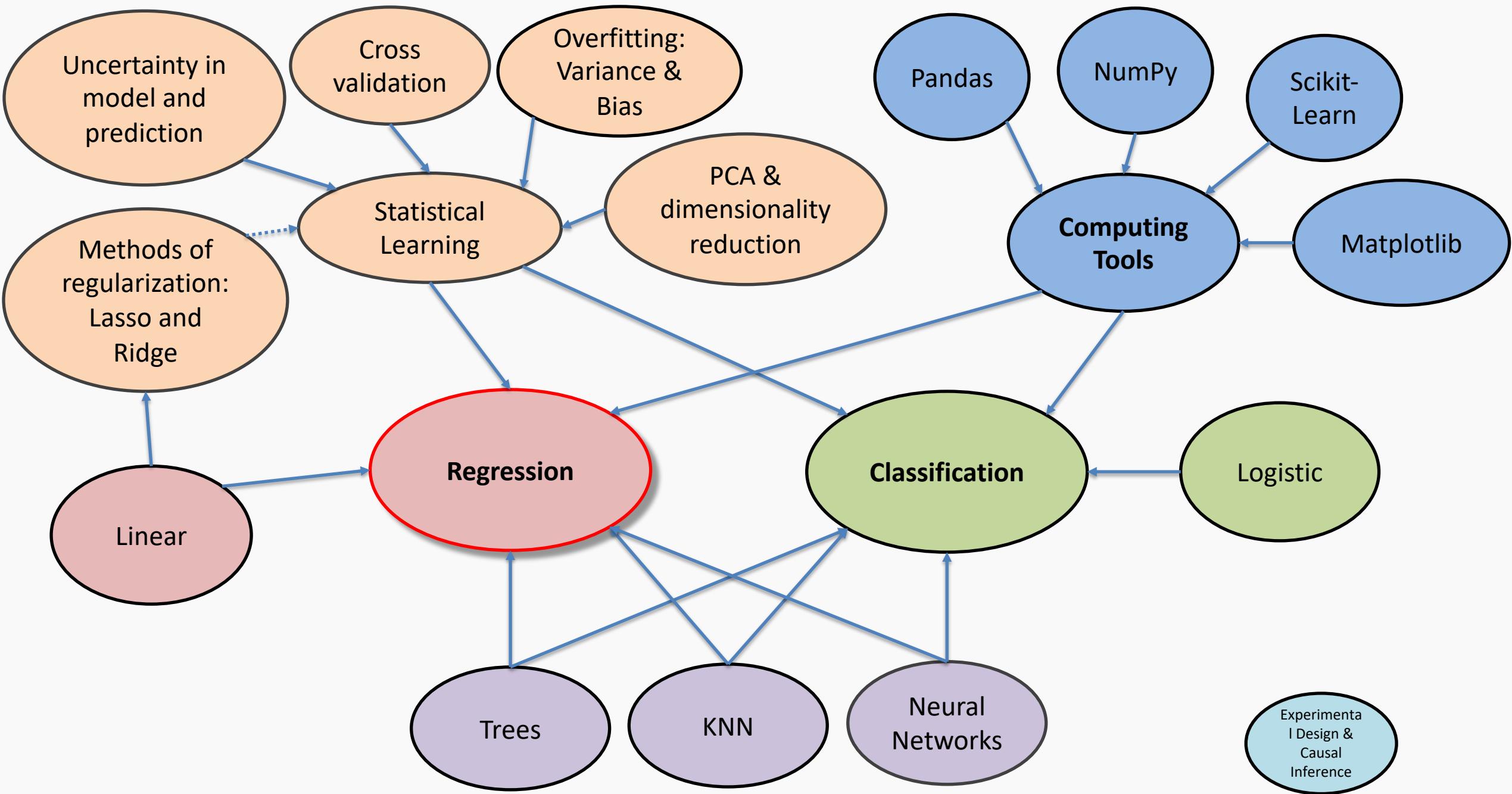


Lecture 24: Review

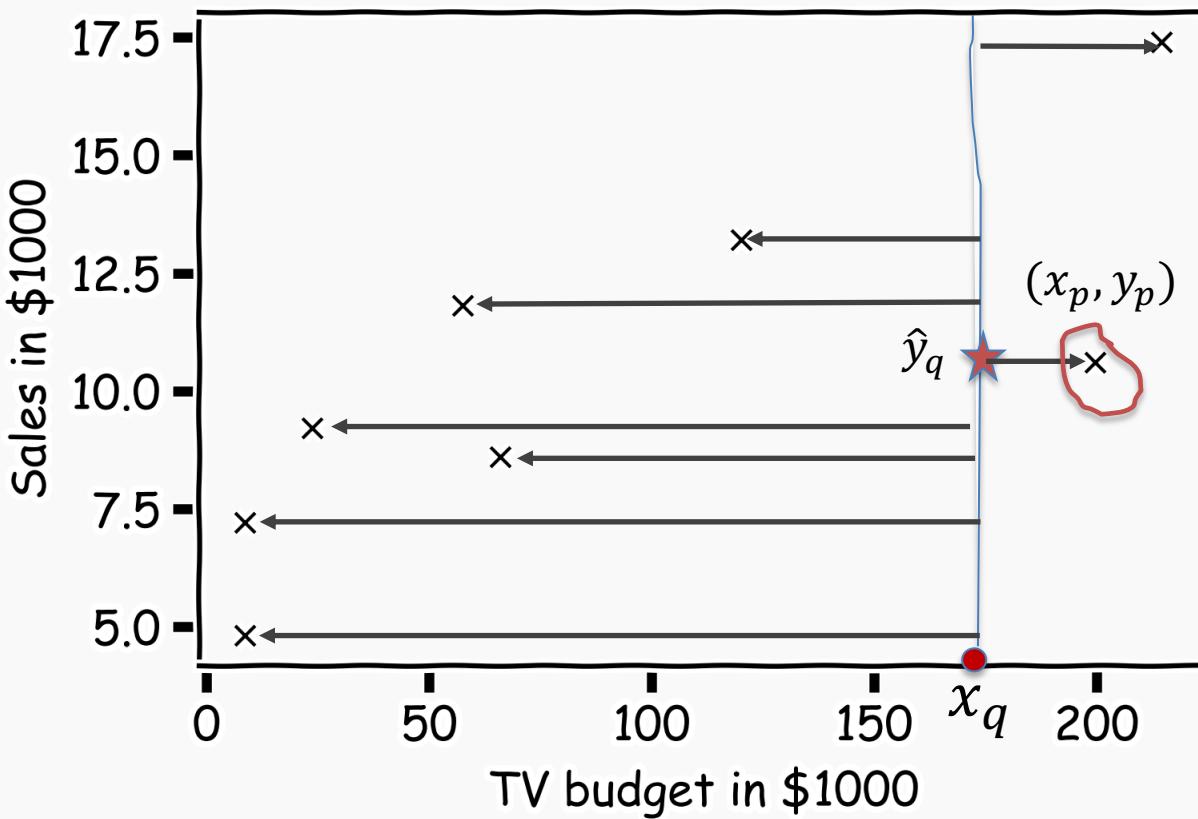
CS109A Introduction to Data Science
Pavlos Protopapas, Kevin Rader and Chris Tanner







Simple Prediction Model



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

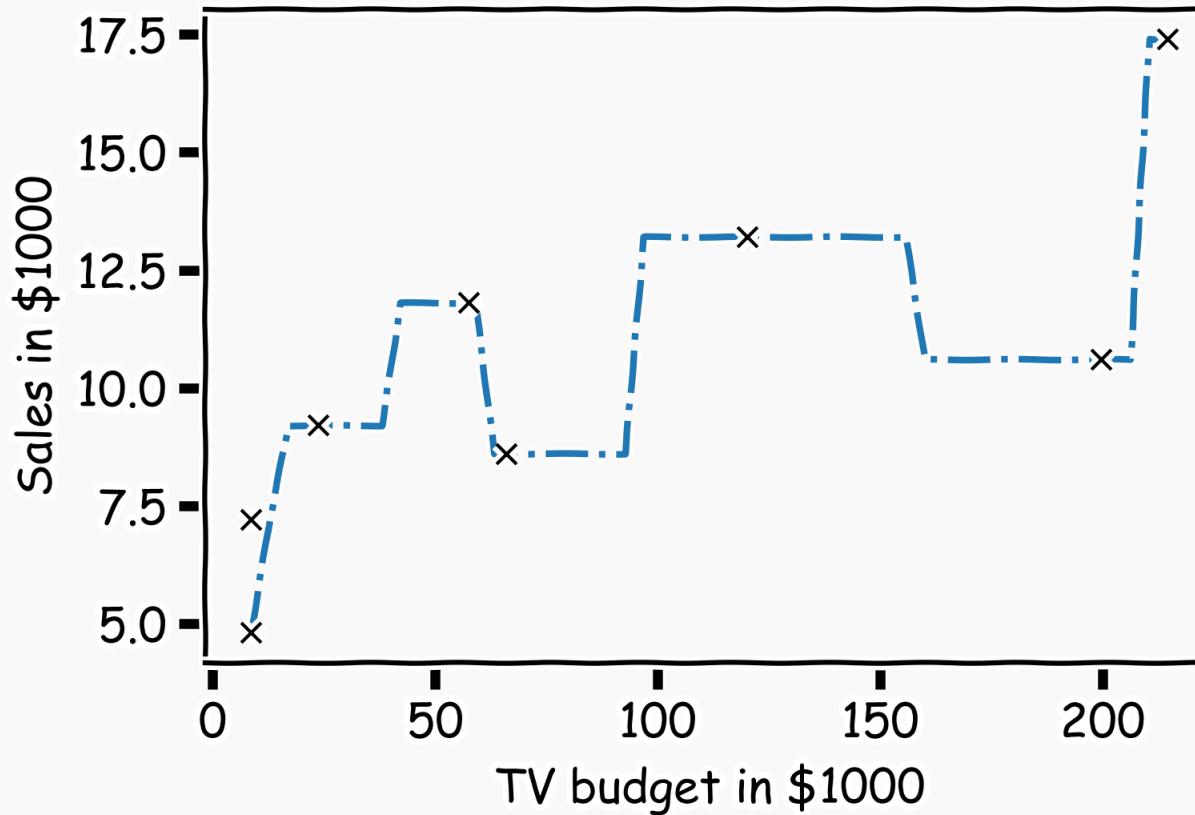
Find distances to all other points
 $D(x_q, x_i)$

Find the nearest neighbor, (x_p, y_p)

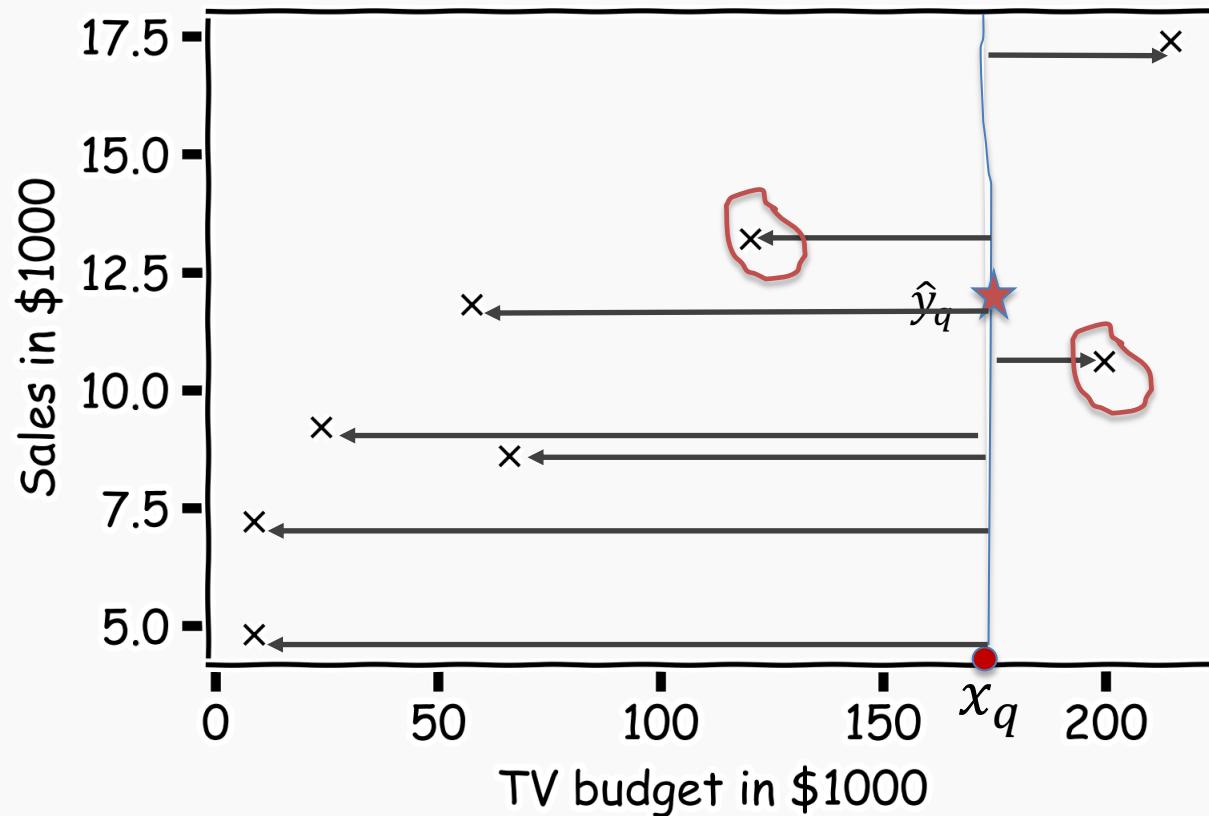
Predict $\hat{y}_q = y_p$

Simple Prediction Model

Do the same for “all” x 's



Extend the Prediction Model



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

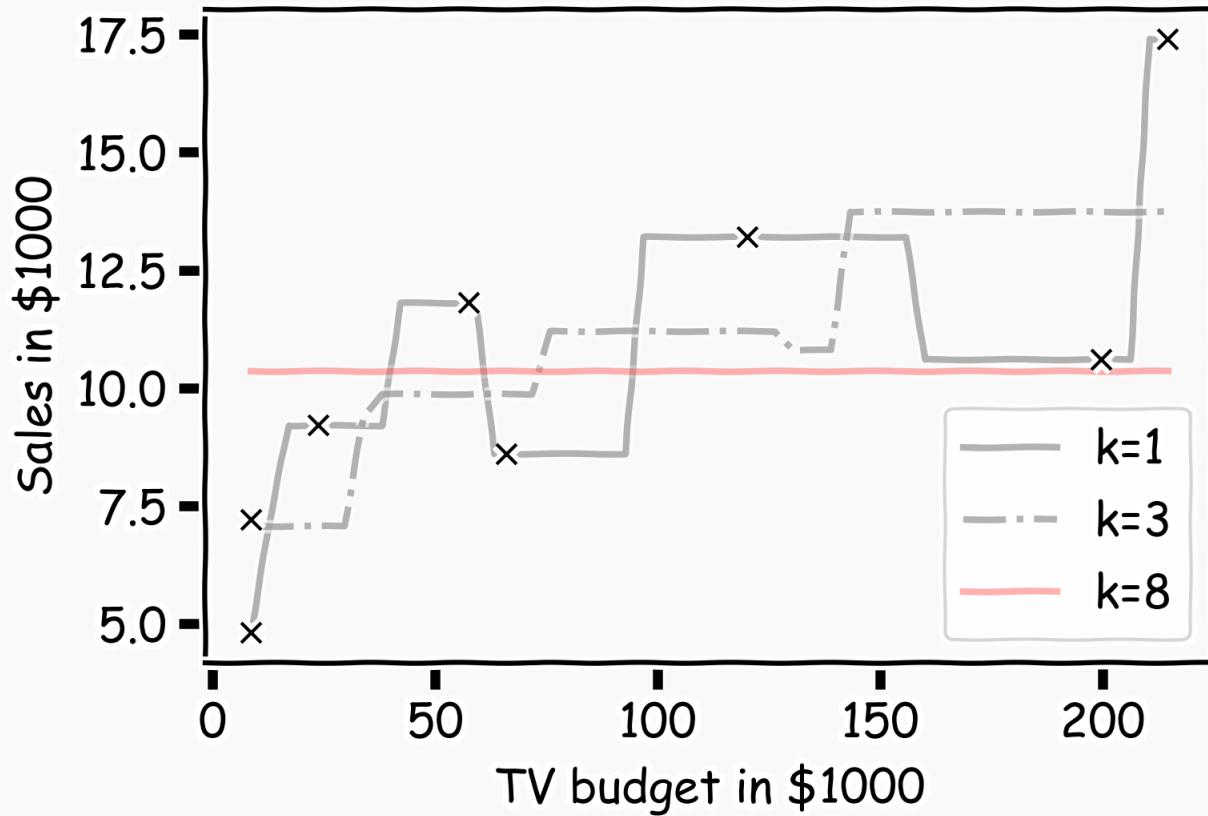
Find distances to all
other points

$$D(x_q, x_i)$$

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

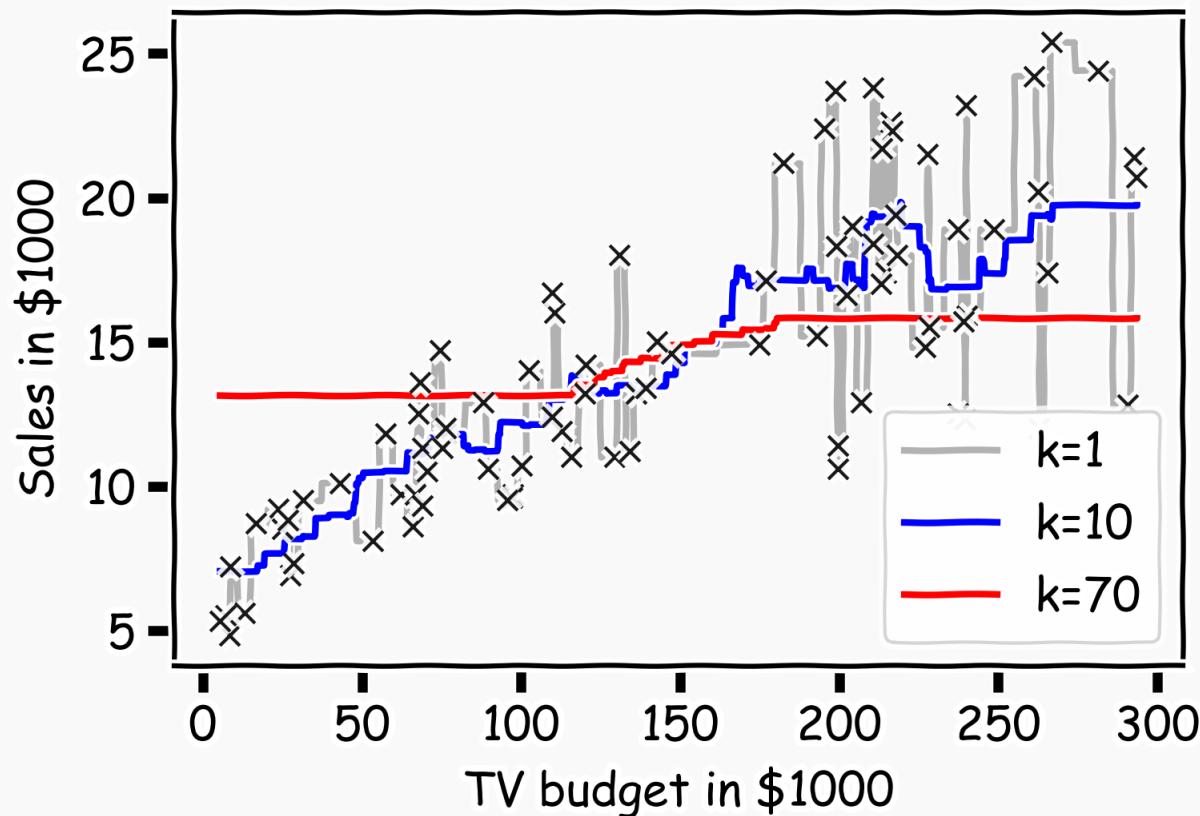
$$\text{Predict } \hat{y}_q = \frac{1}{k} \sum_i^k y_{q_i}$$

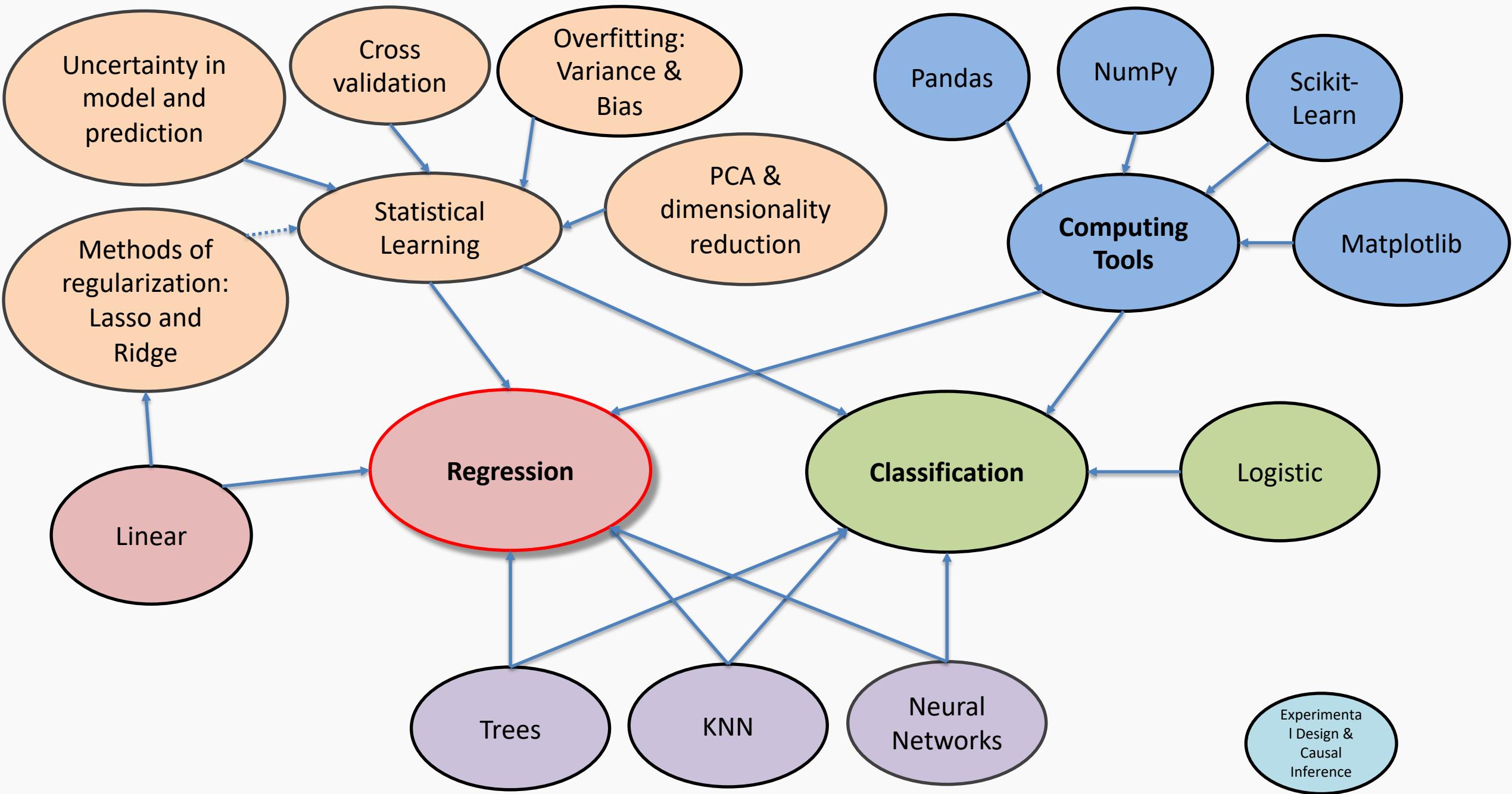
Simple Prediction Models

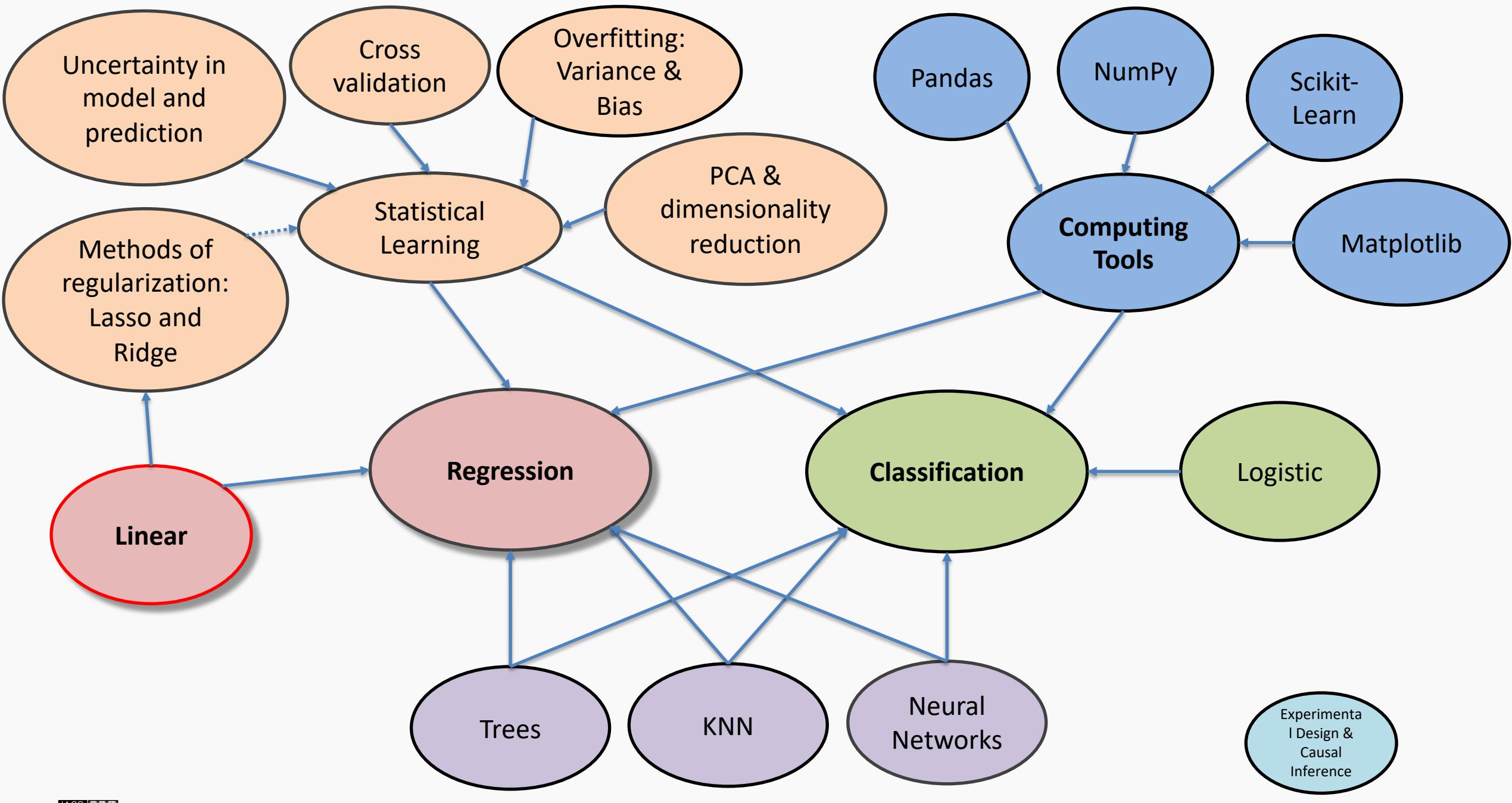


Simple Prediction Models

We can try different k-models on more data

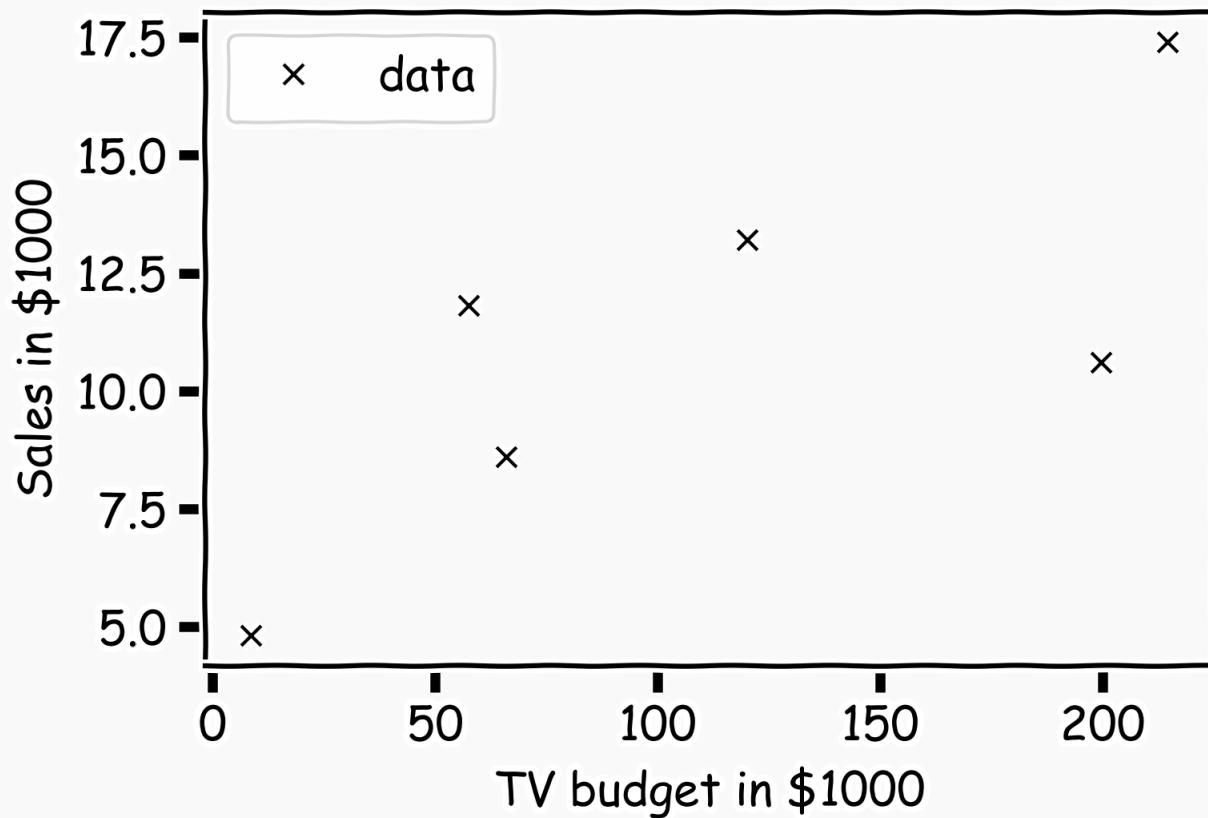






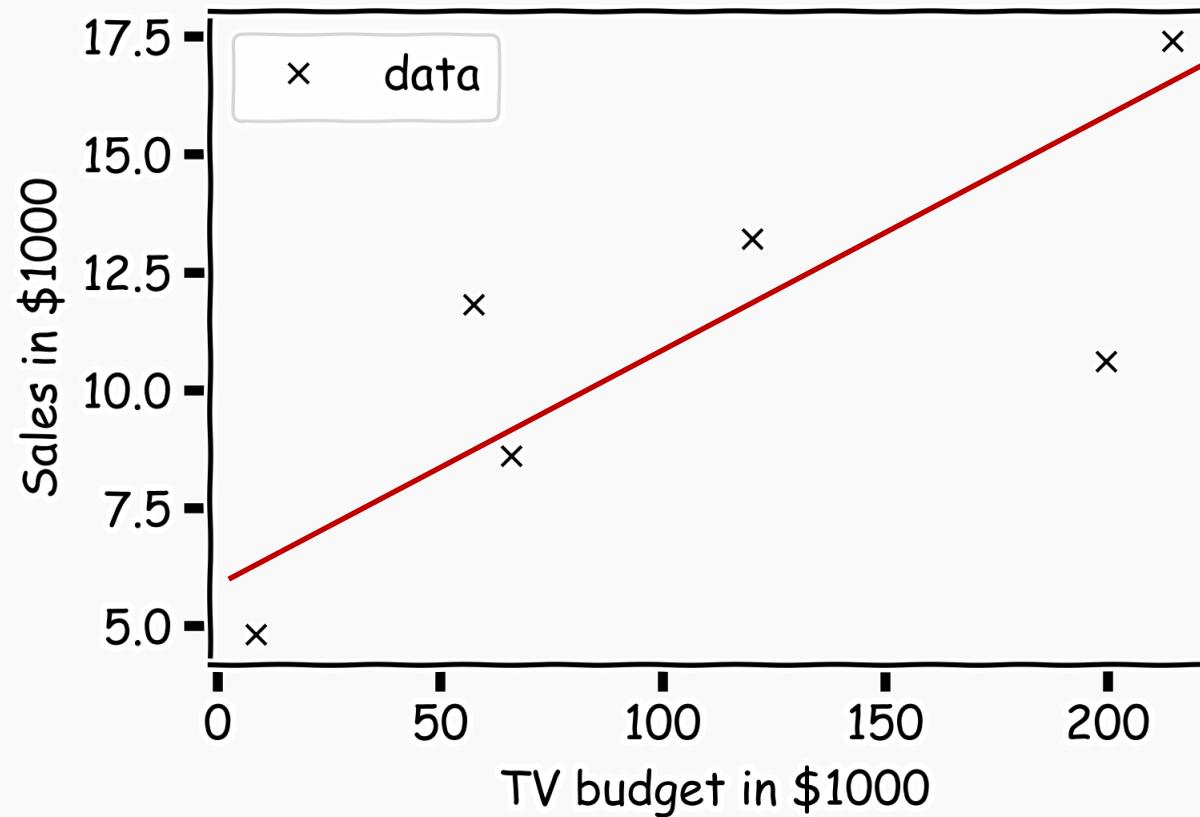
Estimate of the regression coefficients

For a given data set



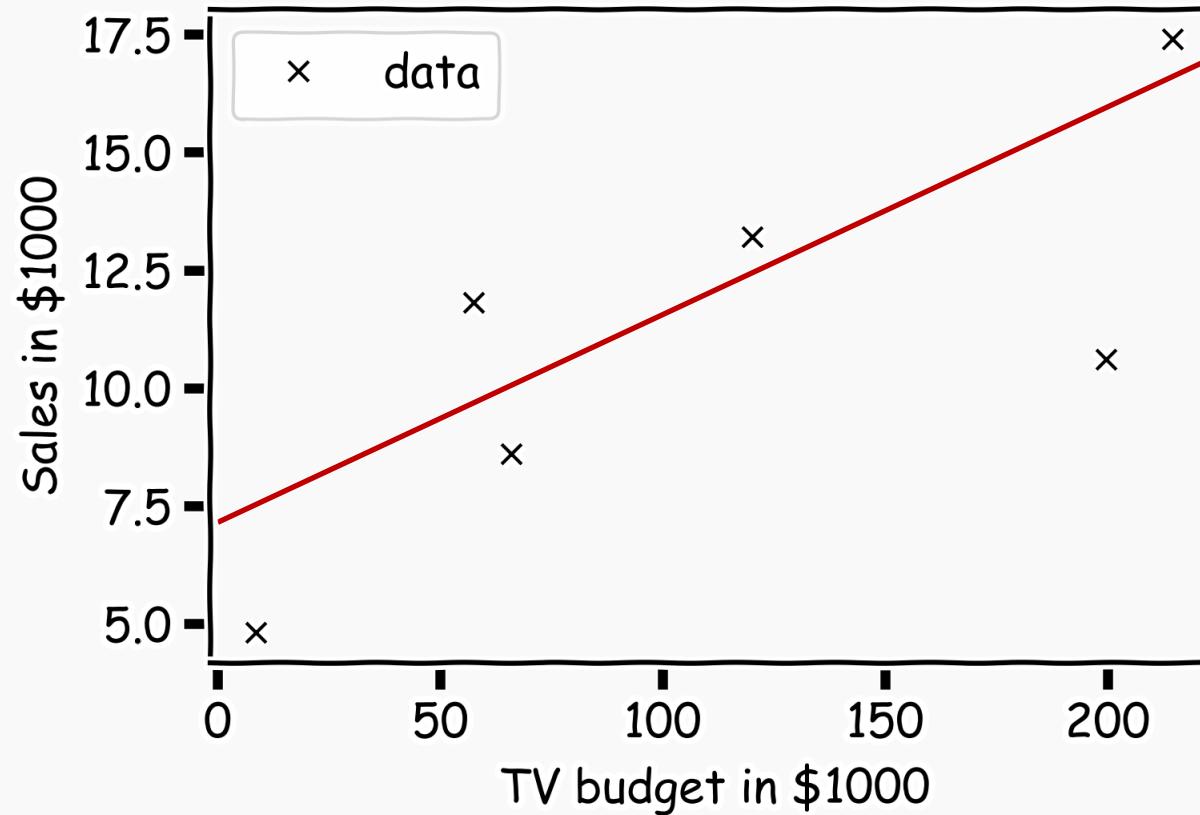
Estimate of the regression coefficients (cont)

Is this line good?



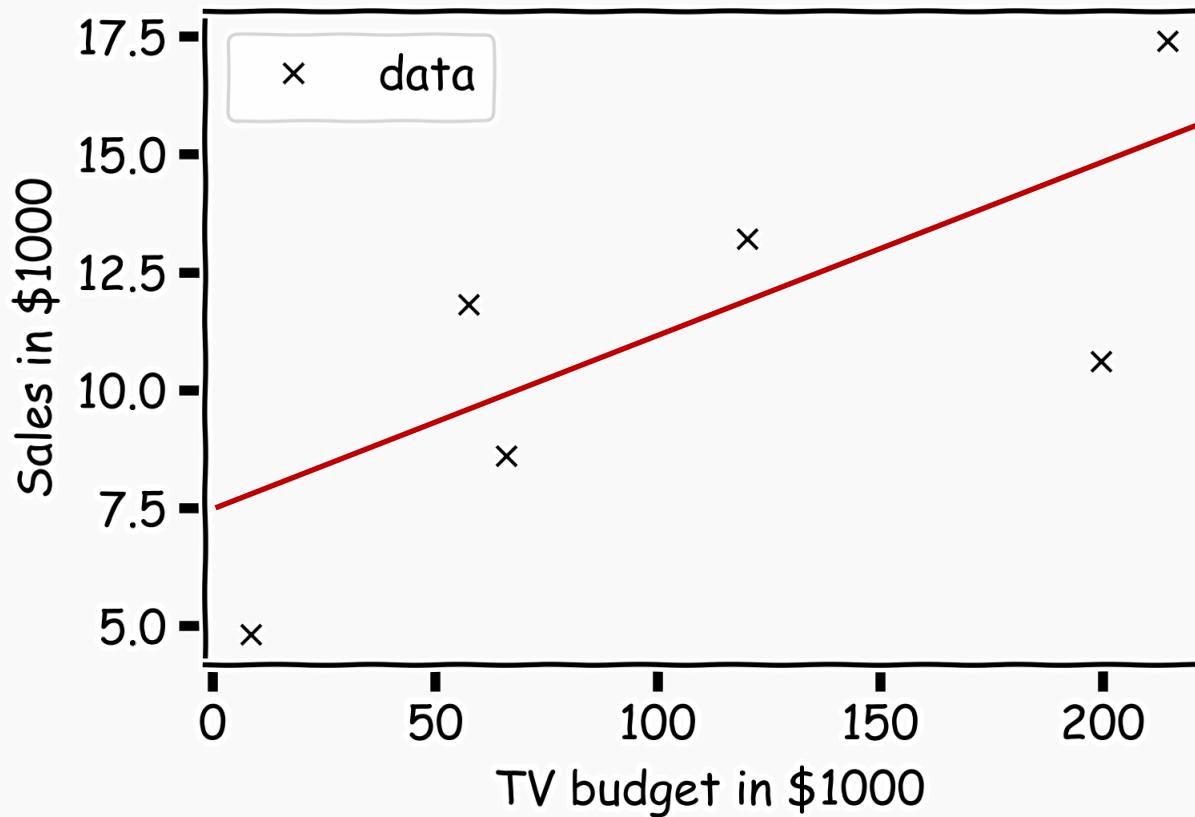
Estimate of the regression coefficients (cont)

Maybe this one?



Estimate of the regression coefficients (cont)

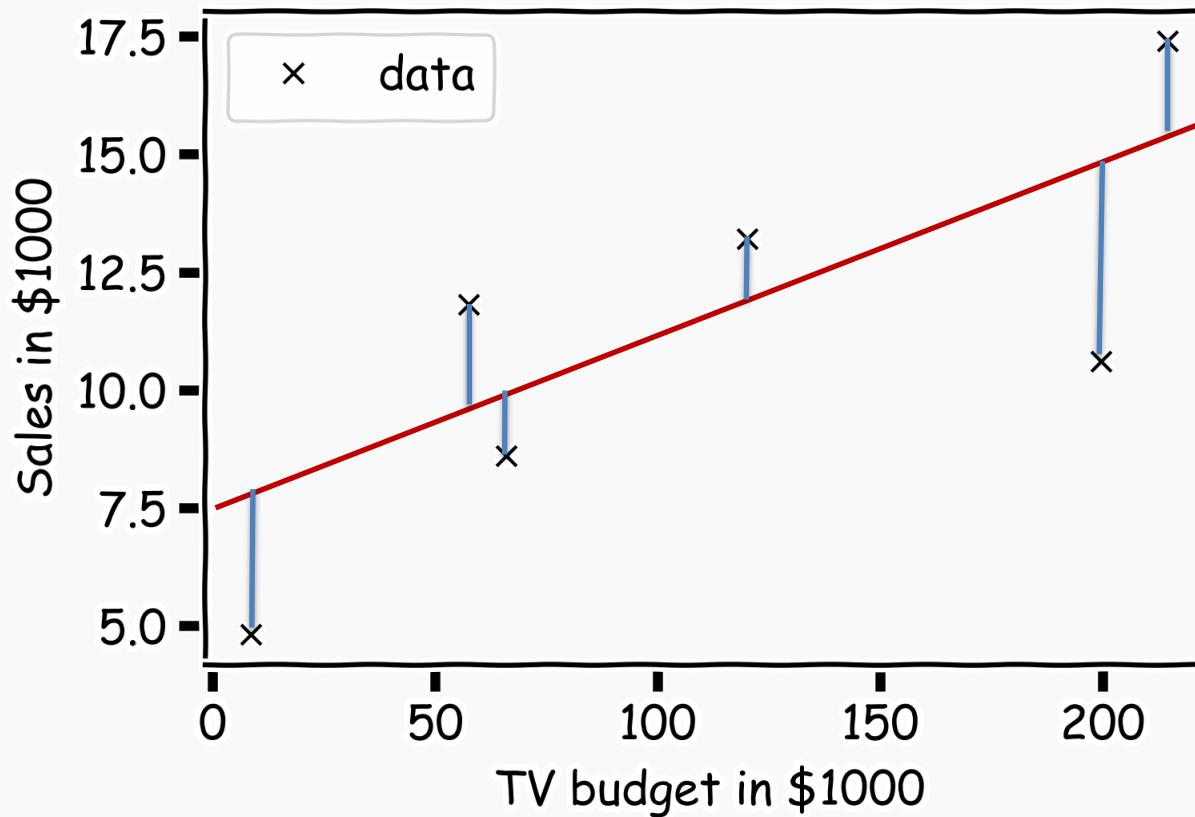
Or this one?



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:

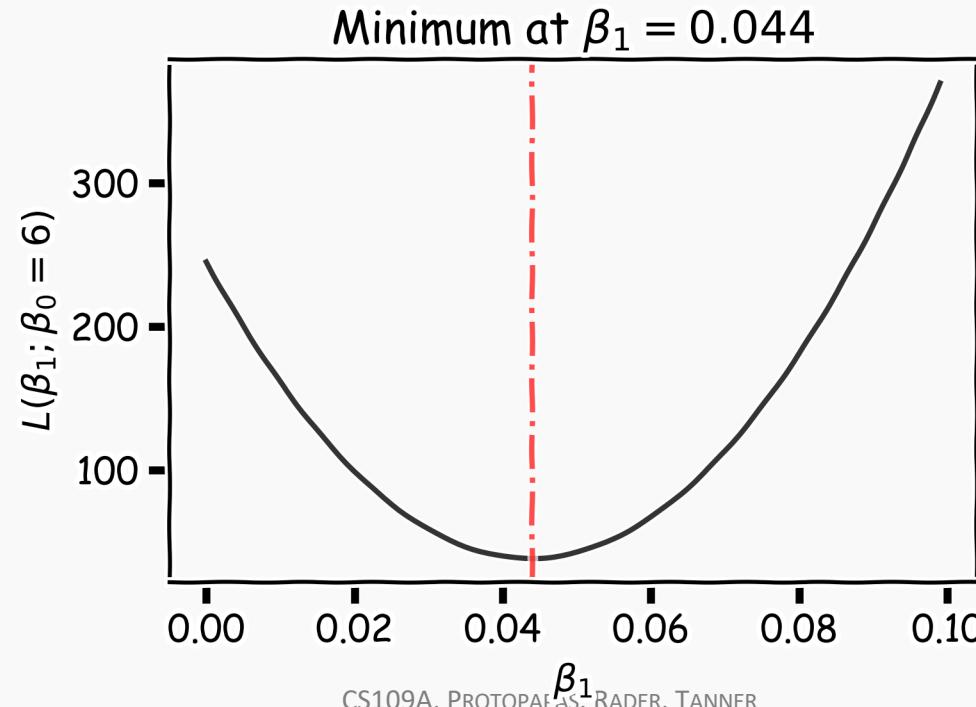
$$\hat{\beta}_0, \hat{\beta}_1 = \operatorname{argmin}_{\beta_0, \beta_1} L(\beta_0, \beta_1).$$



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 $\hat{\beta}_1$:

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

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

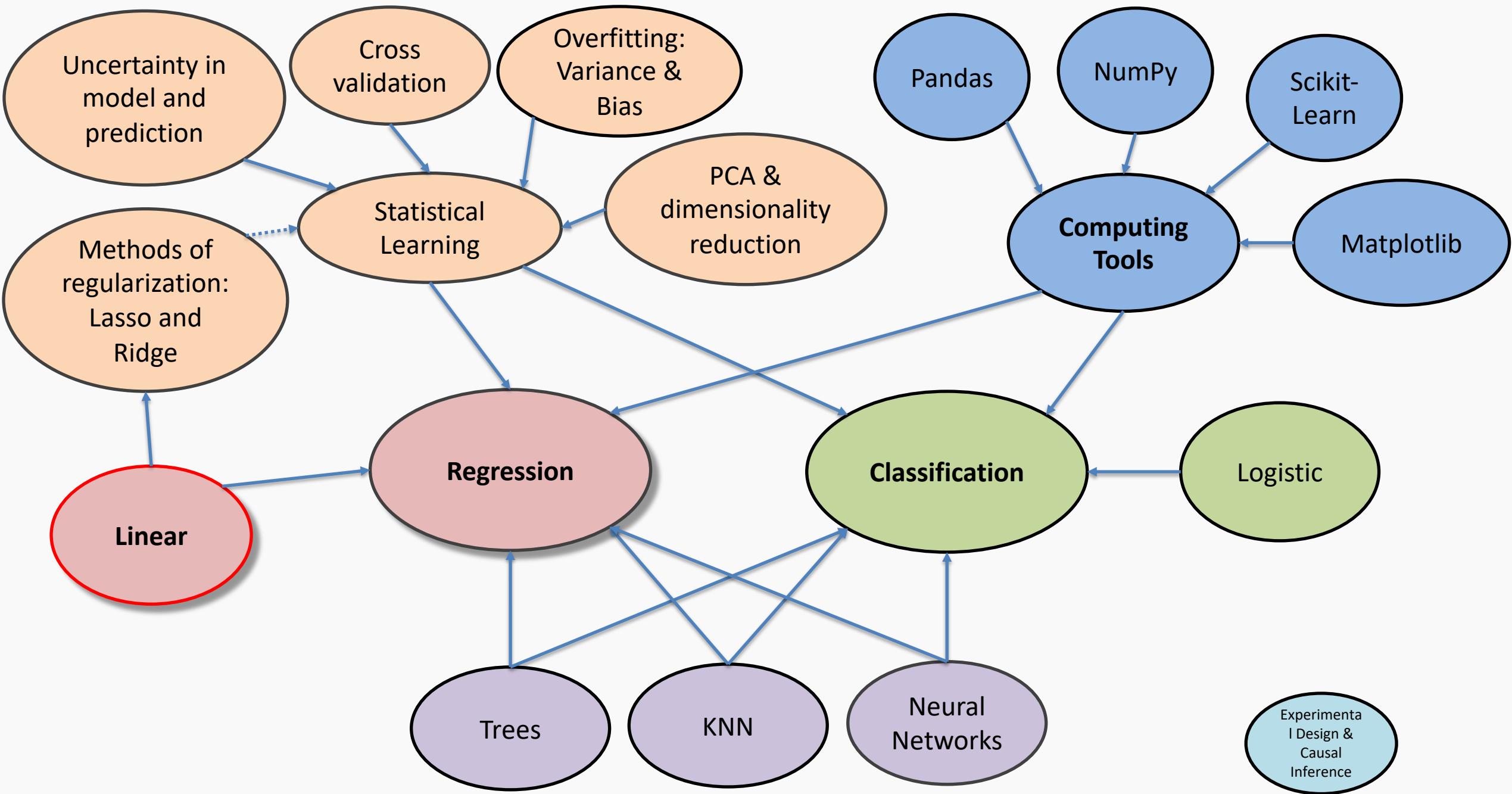
where \bar{y} and \bar{x} are sample means.

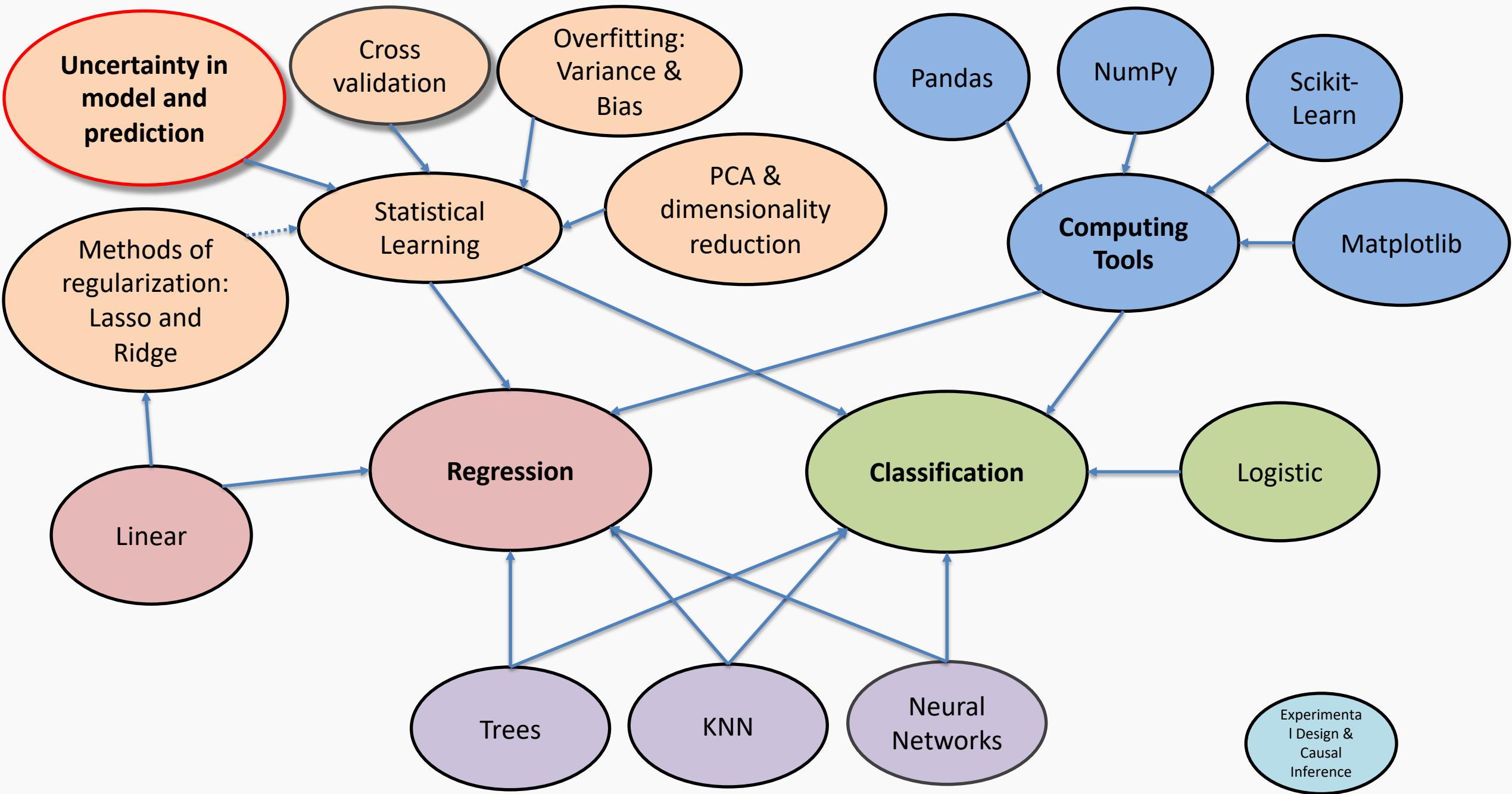
The line:

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

is called the **regression line**.

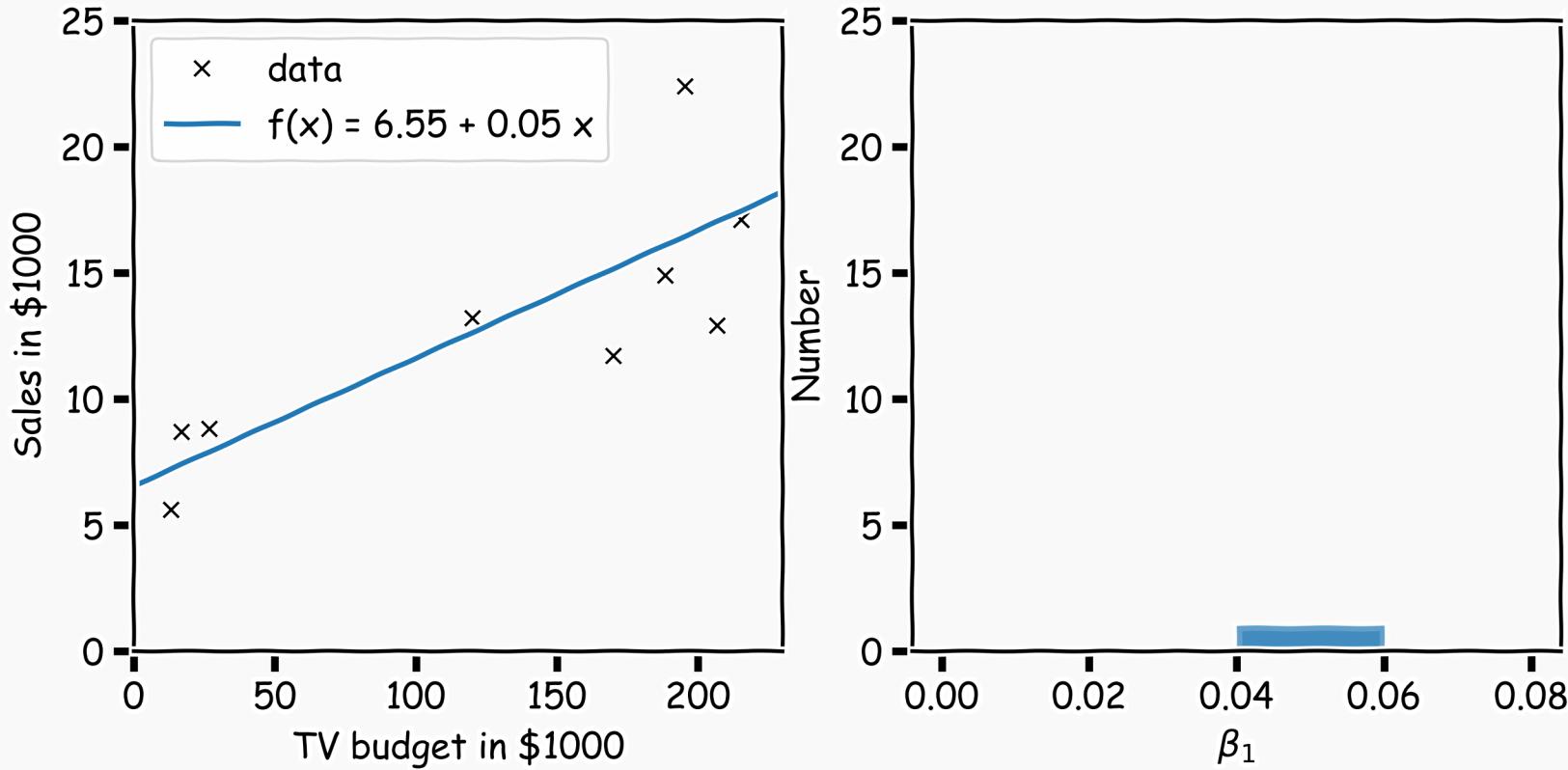






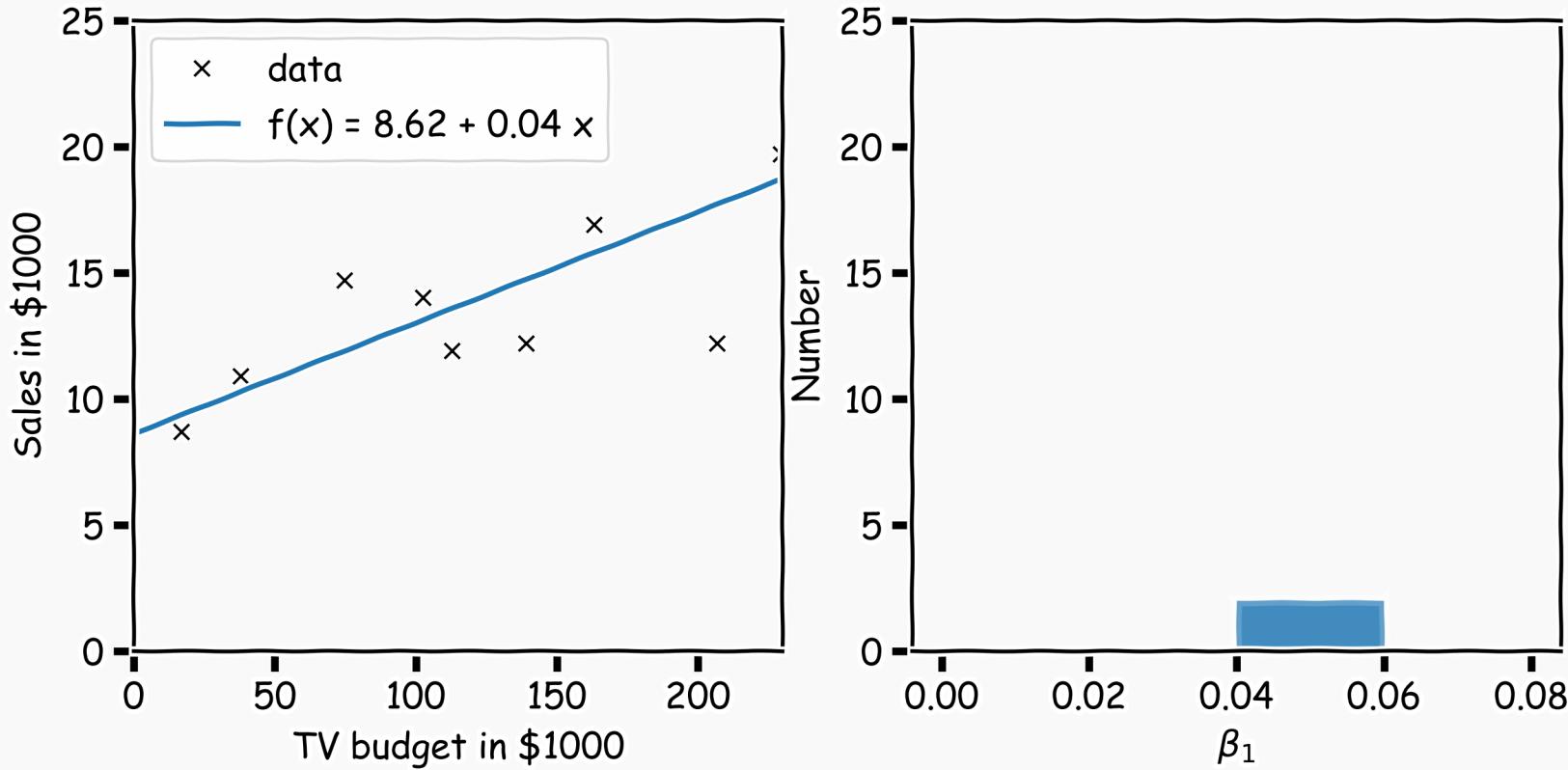
Confidence intervals for the predictors estimates (cont)

In our magical realisms, we can now sample multiple times



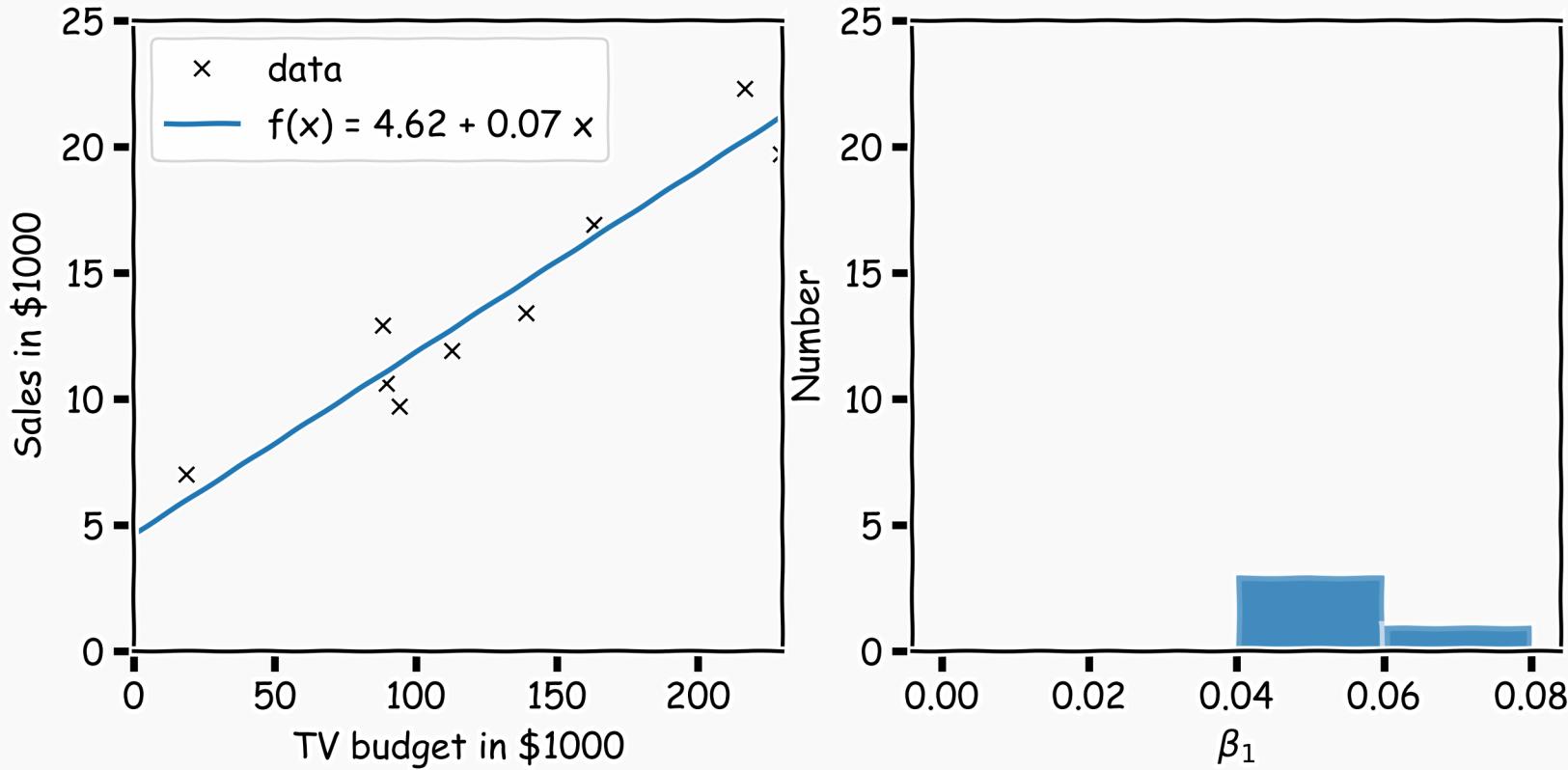
Confidence intervals for the predictors estimates (cont)

Another sample



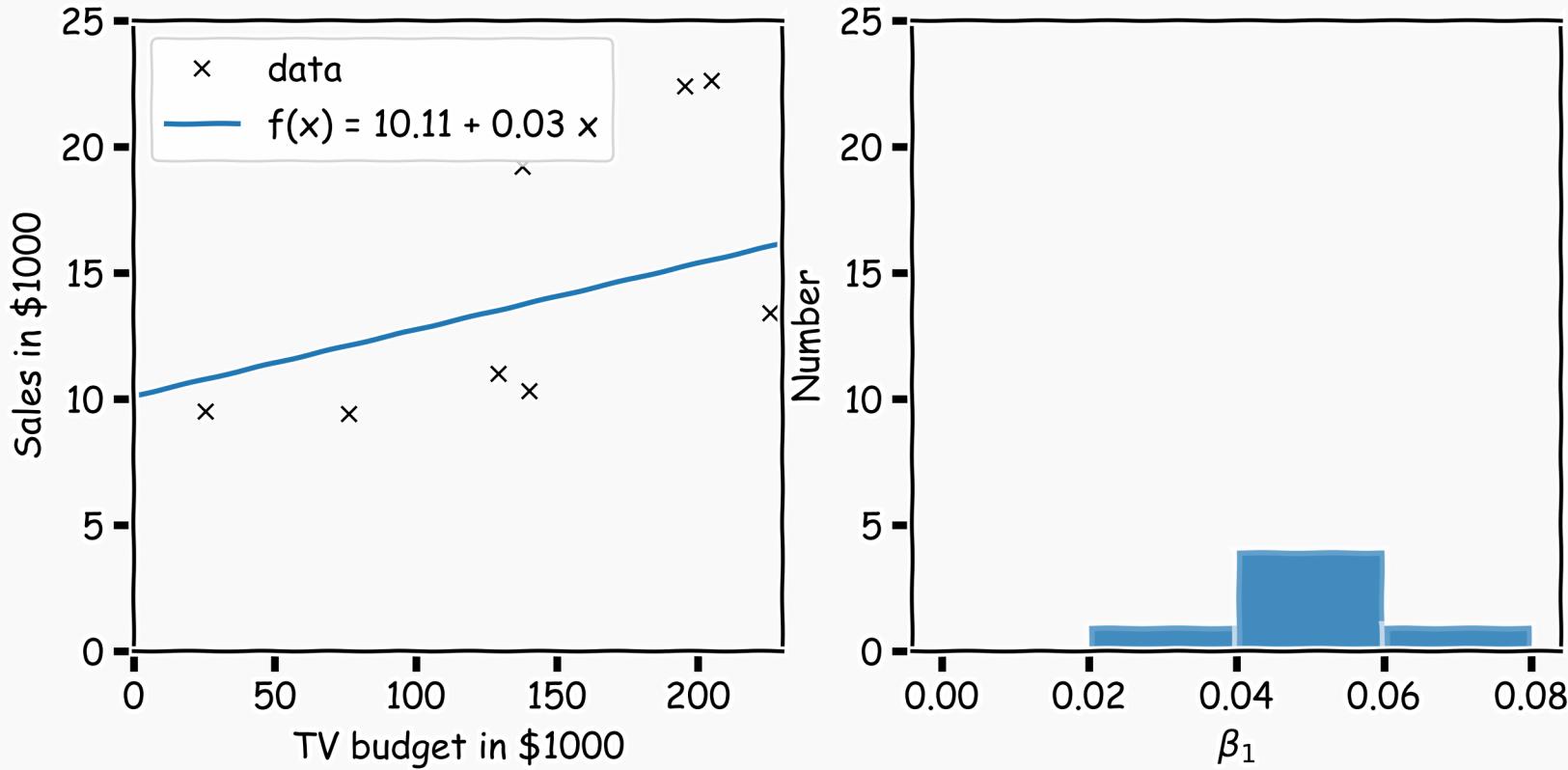
Confidence intervals for the predictors estimates (cont)

Another sample



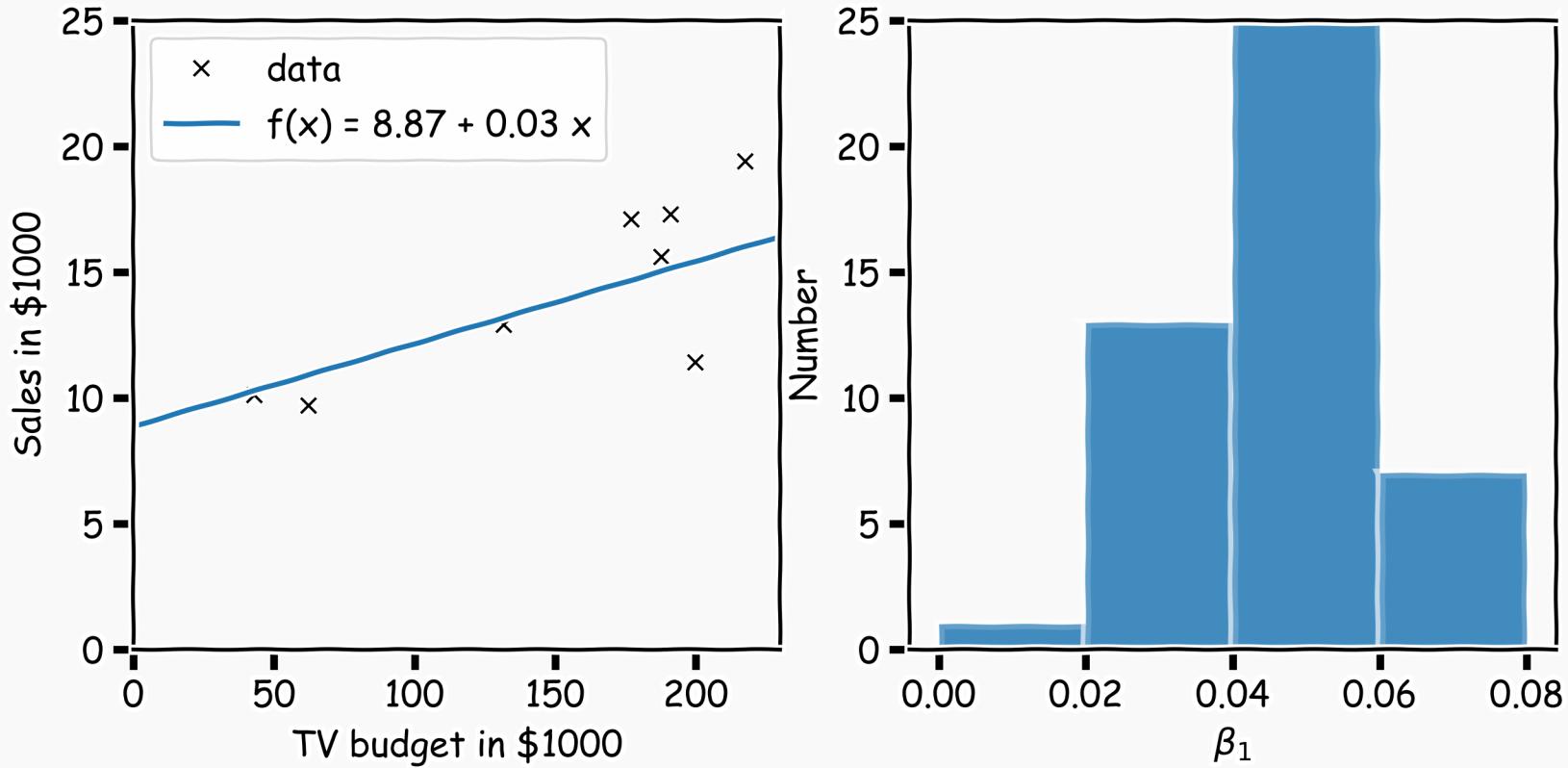
Confidence intervals for the predictors estimates (cont)

And another sample



Confidence intervals for the predictors estimates (cont)

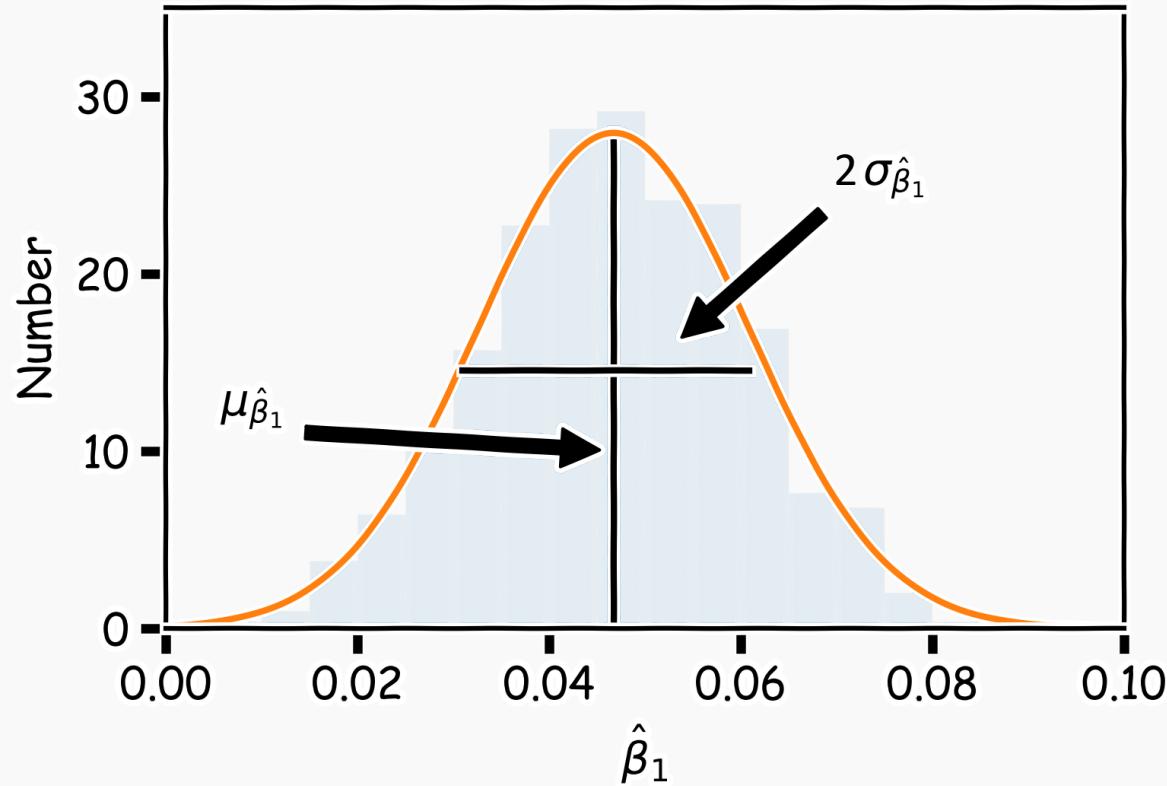
Repeat this for 100 times



Confidence intervals for the predictors estimates (cont)

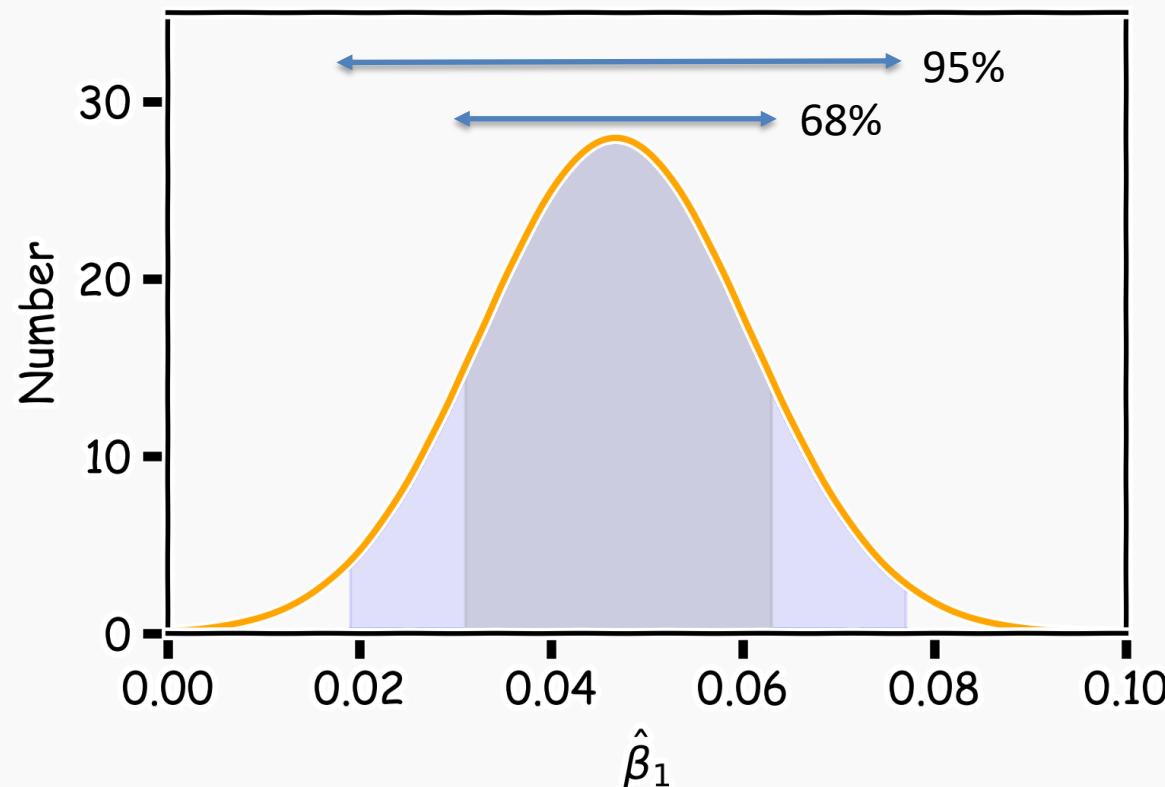
We can now estimate the mean and standard deviation of all the estimates $\hat{\beta}_1$.

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)

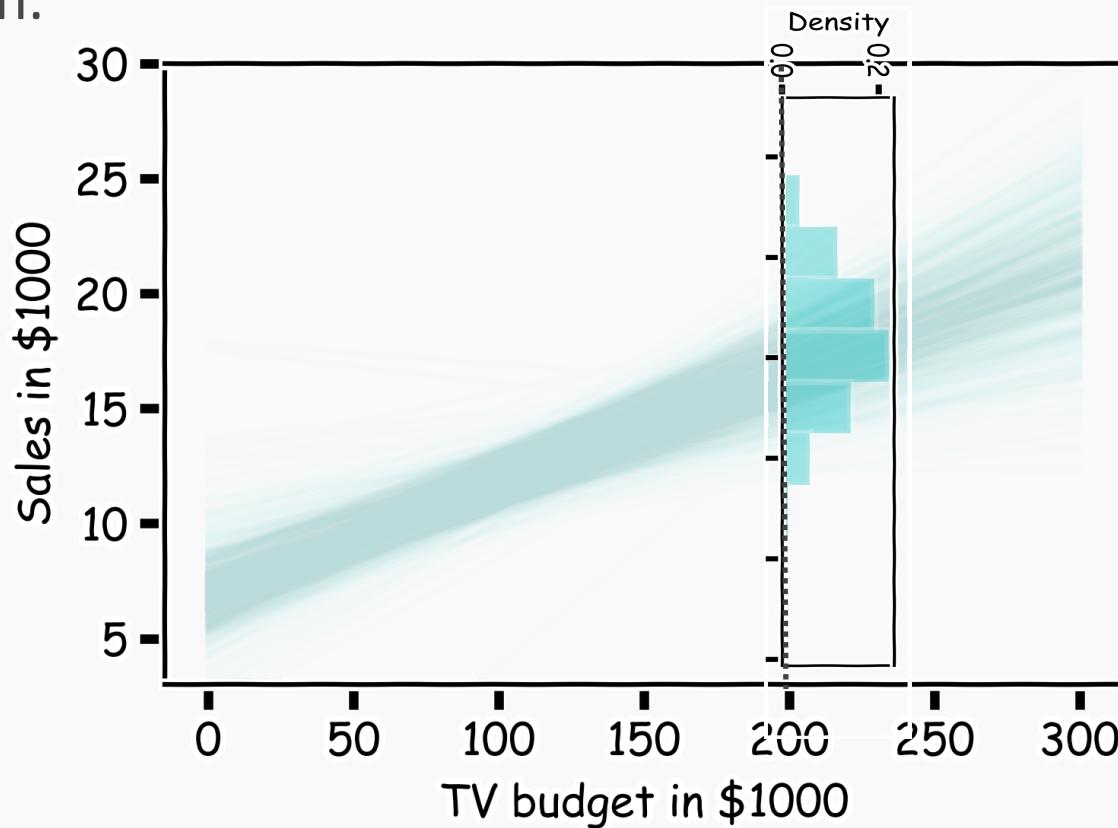
Finally we can calculate the confidence intervals, which are the ranges of values such that the **true** value of β_1 is contained in this interval with n percent probability.



How well do we know \hat{f} ?

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

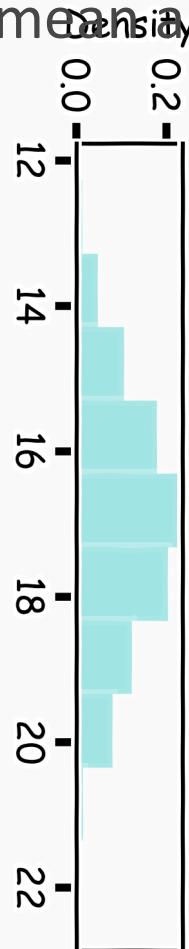
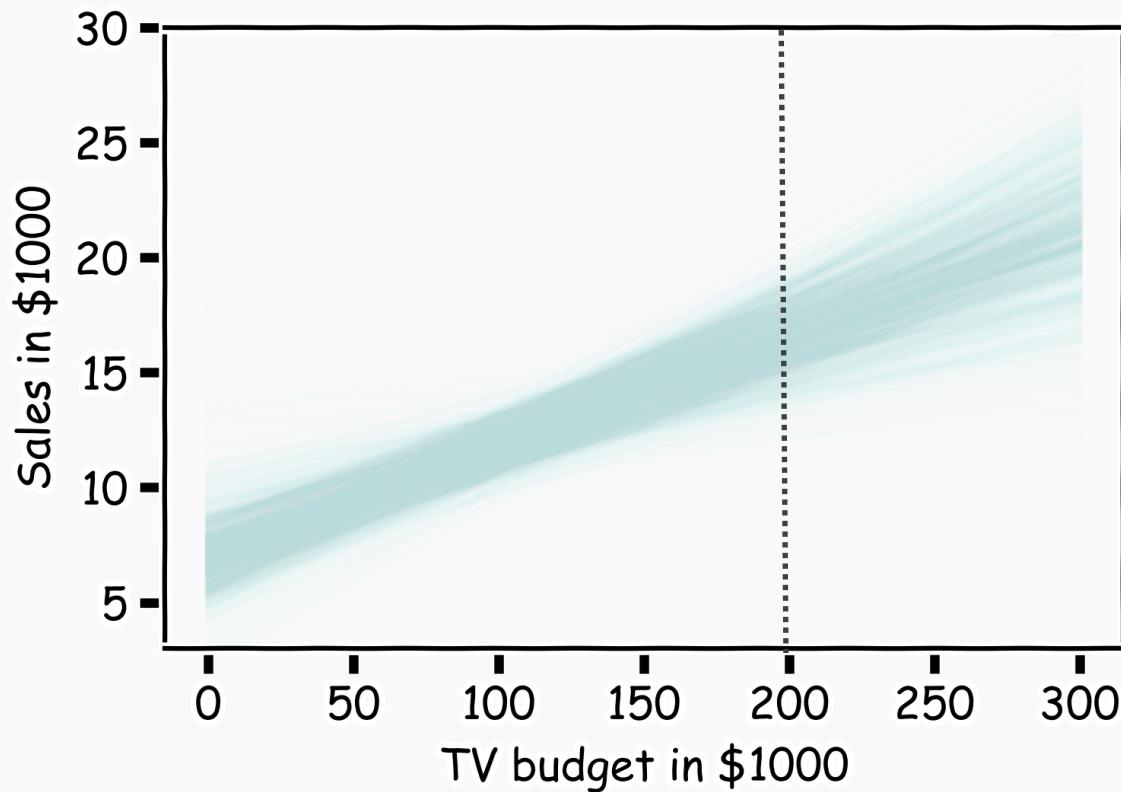
For a given x , we examine the distribution of \hat{f} , and determine the mean and standard deviation.



How well do we know \hat{f} ?

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

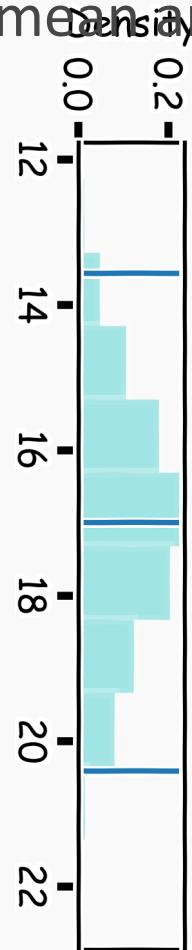
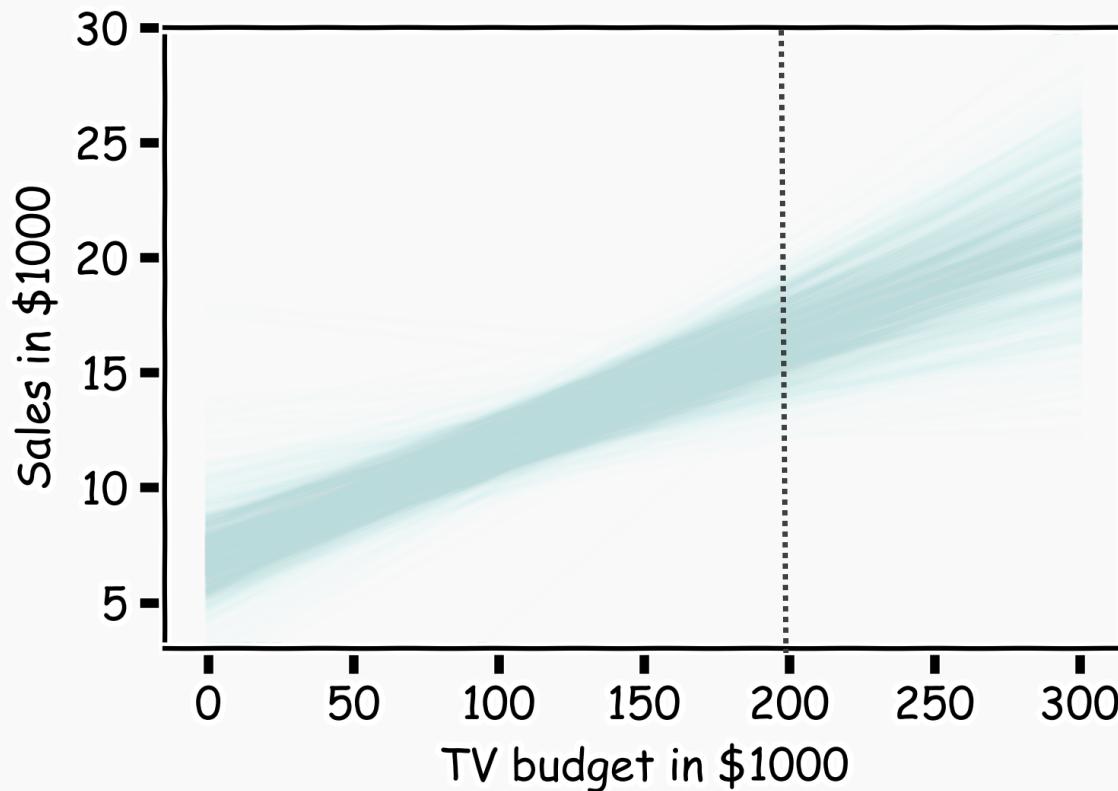
For a given x , we examine the distribution of \hat{f} , and determine the mean and standard deviation.



How well do we know \hat{f} ?

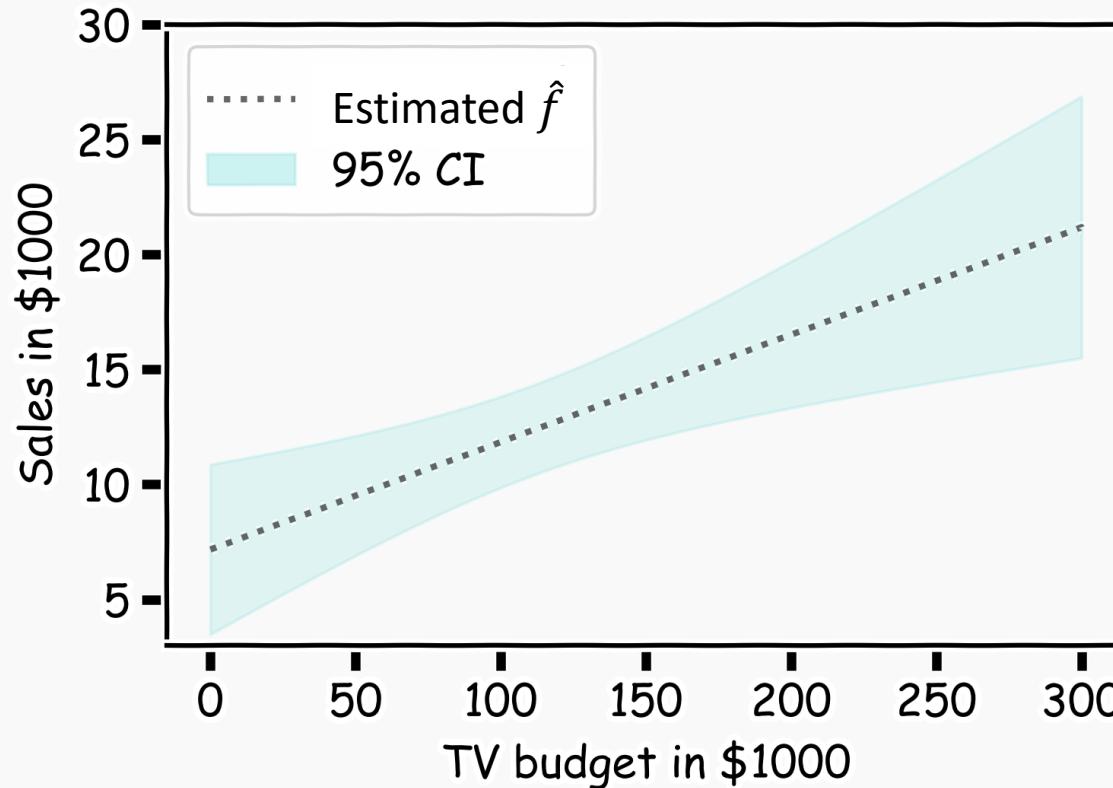
Below we show all regression lines for a thousand of such sub-samples.

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

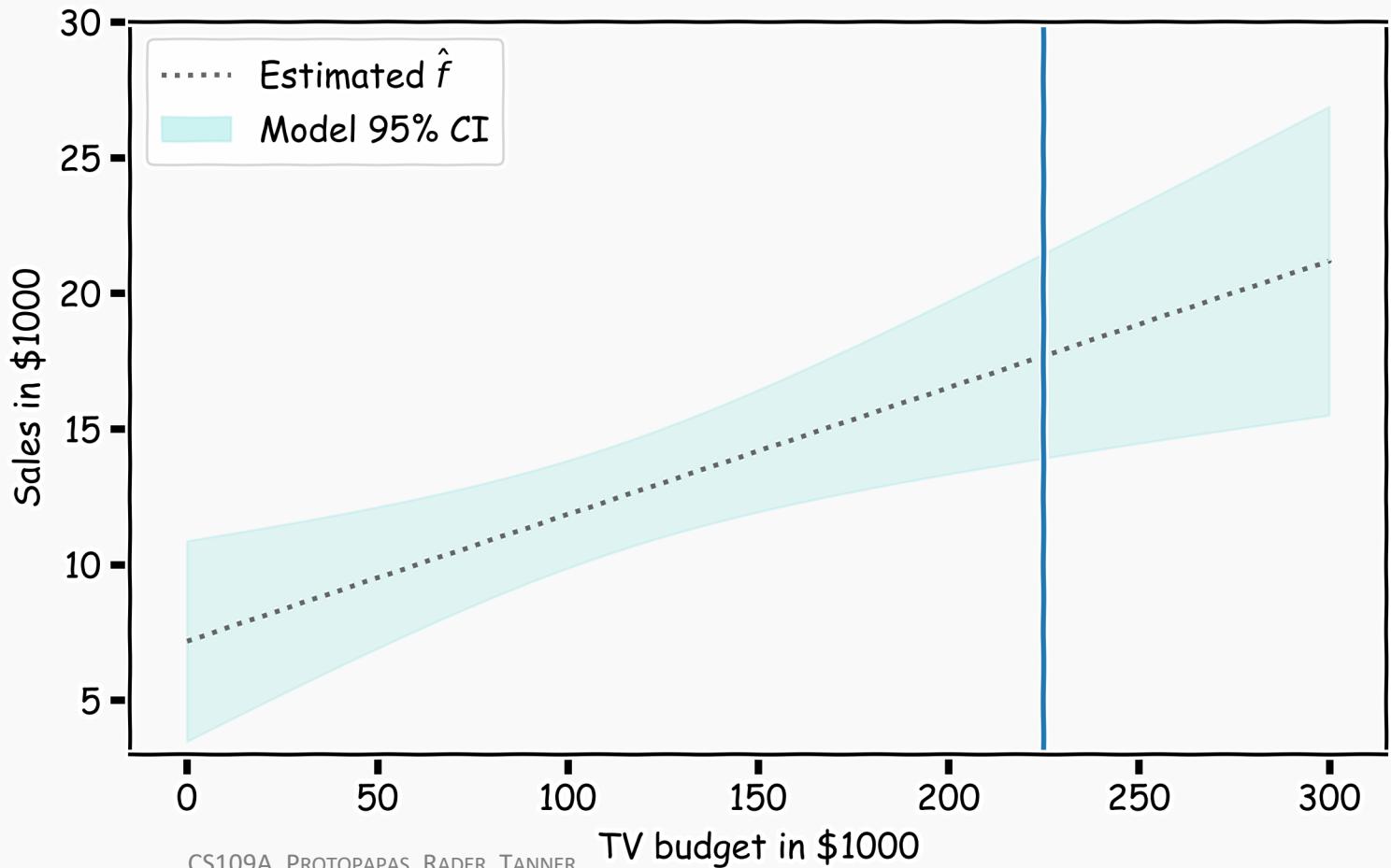


How well do we know \hat{f} ?

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).

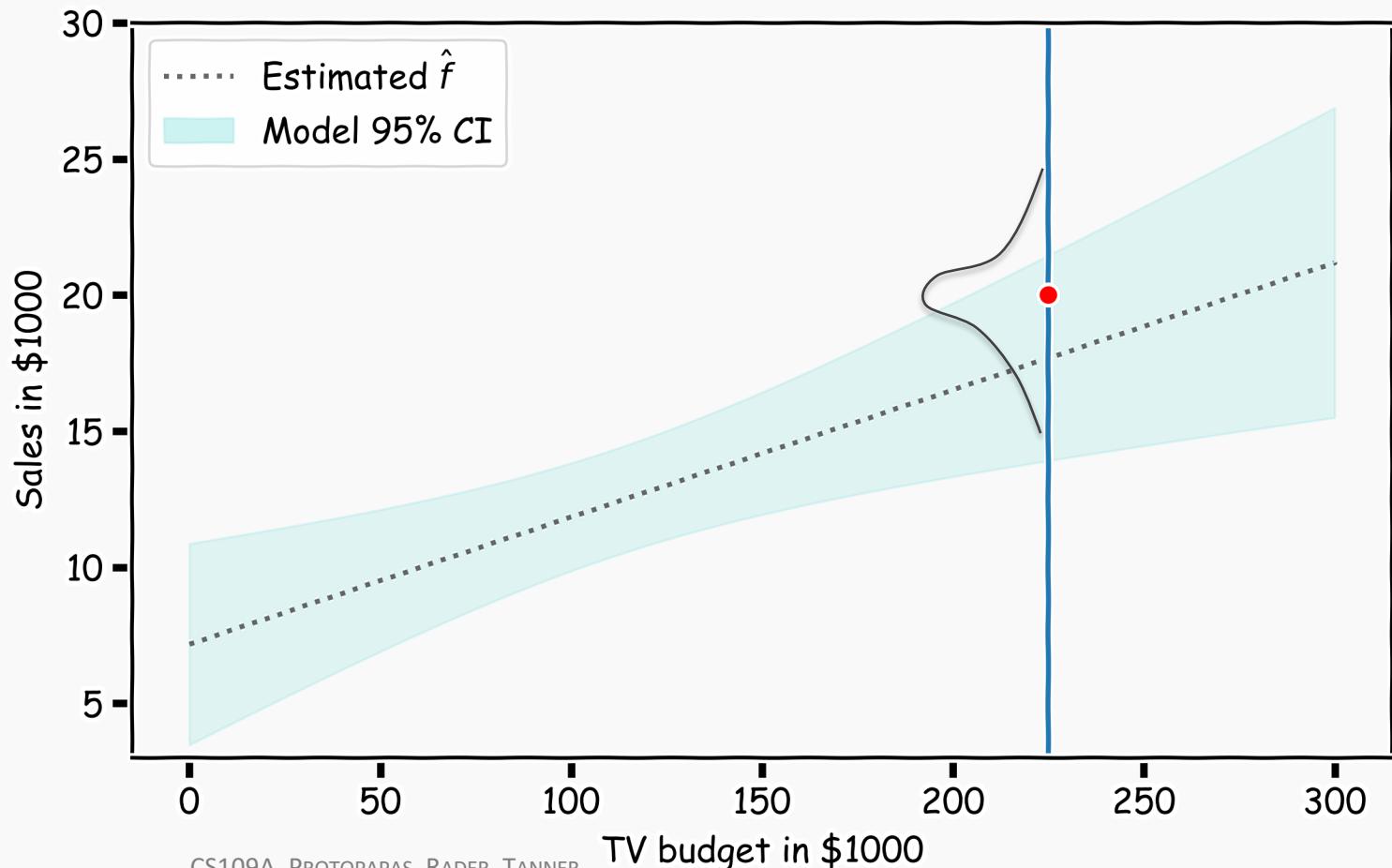


Confidence in predicting \hat{y}



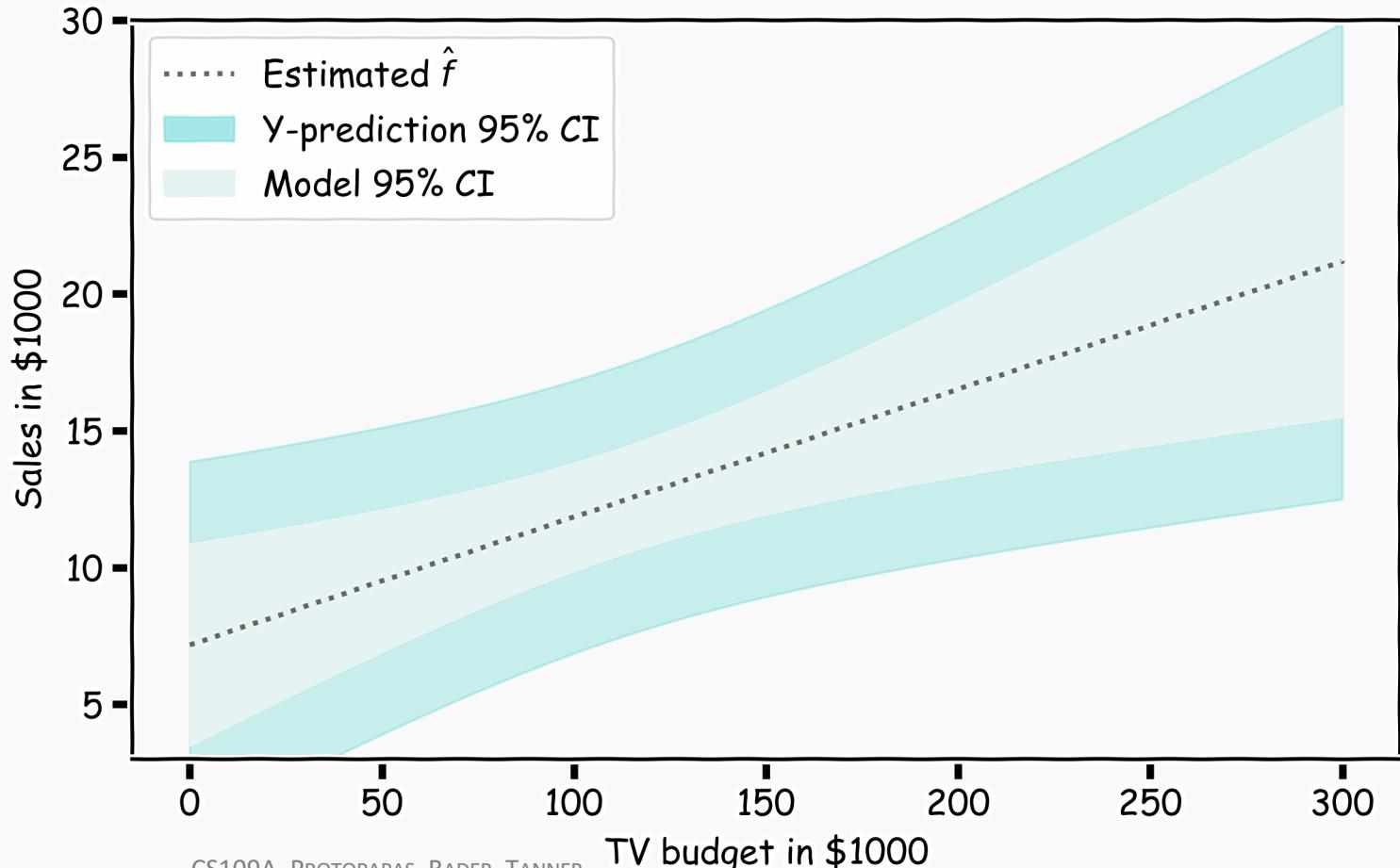
Confidence in predicting \hat{y}

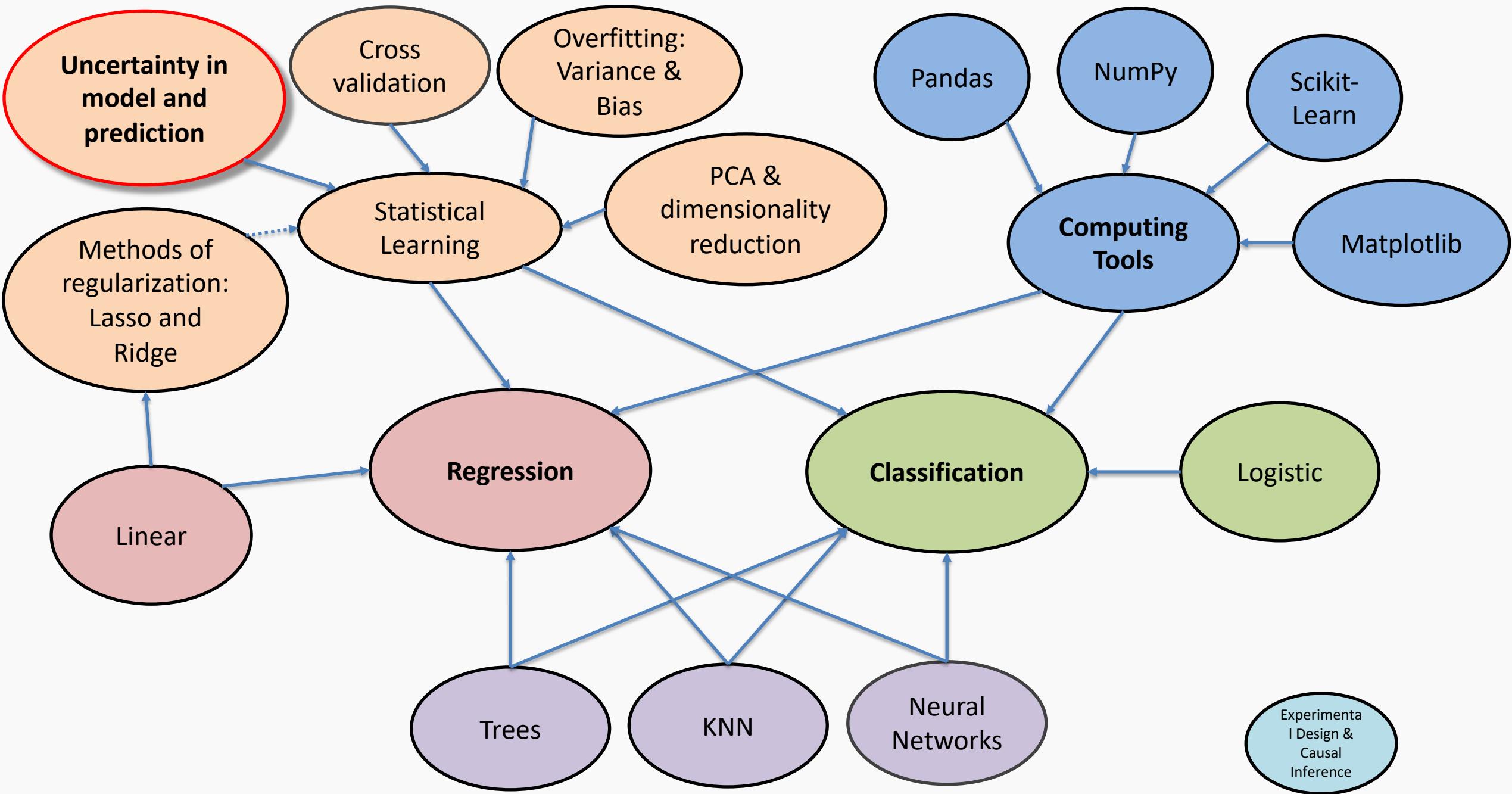
- for a given x , we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f, \sigma_\epsilon)$

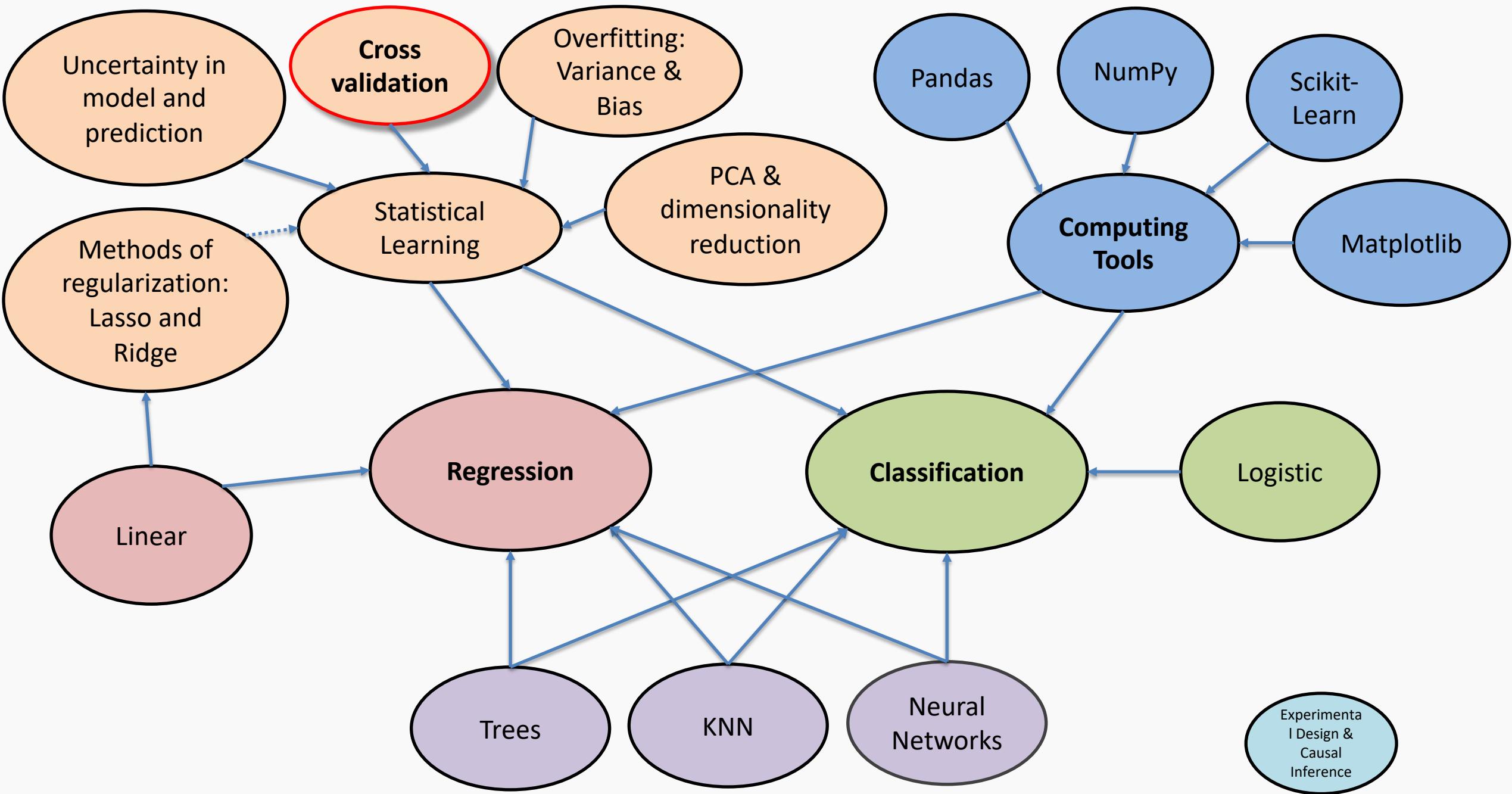


Confidence in predicting \hat{y}

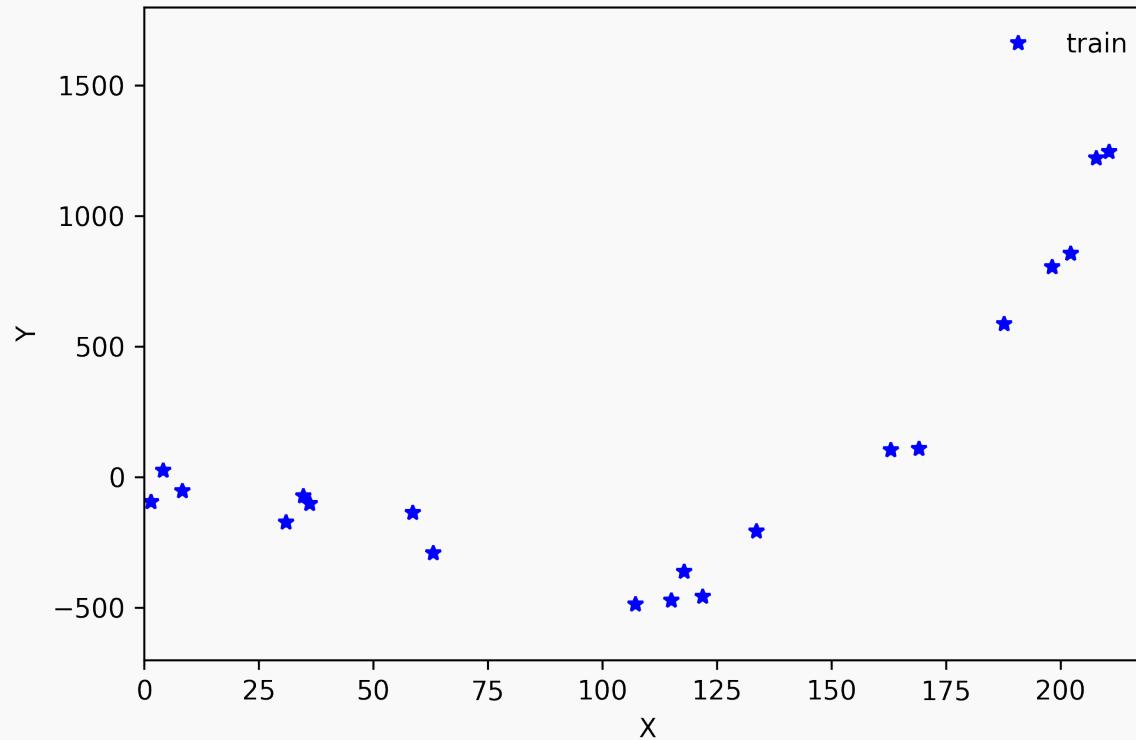
- for a given x , we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f, \sigma_\epsilon)$
- The prediction confidence intervals are then



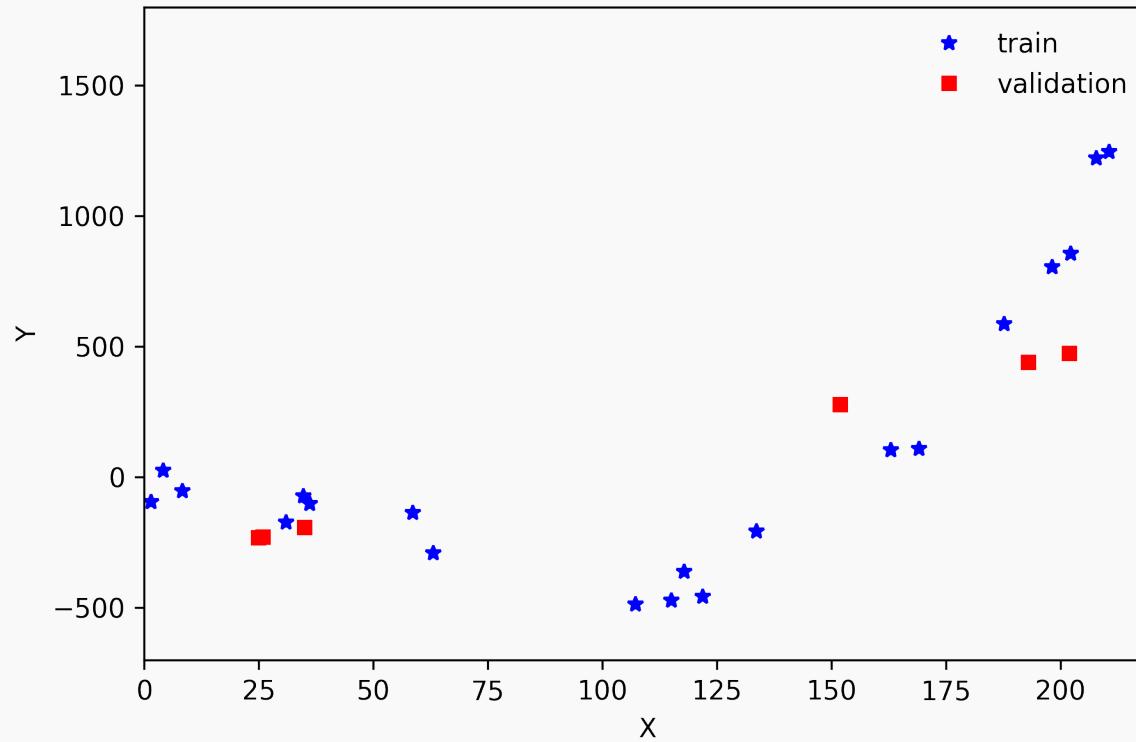




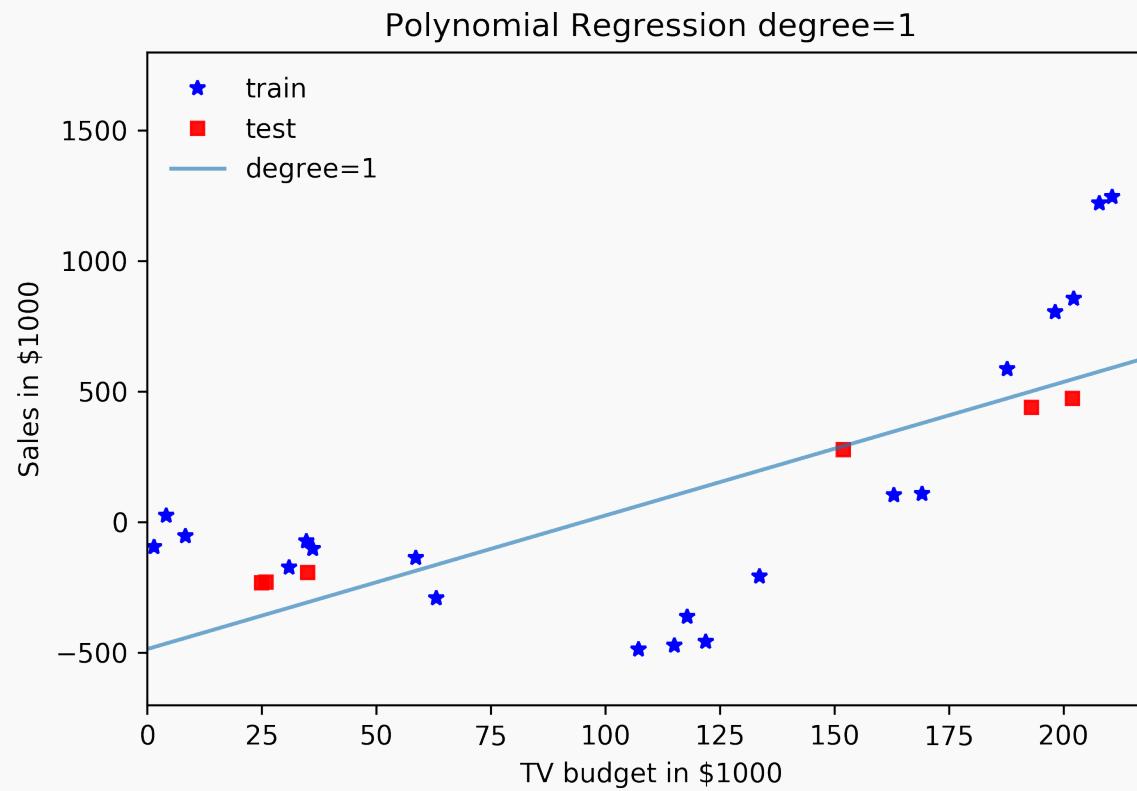
Cross Validation



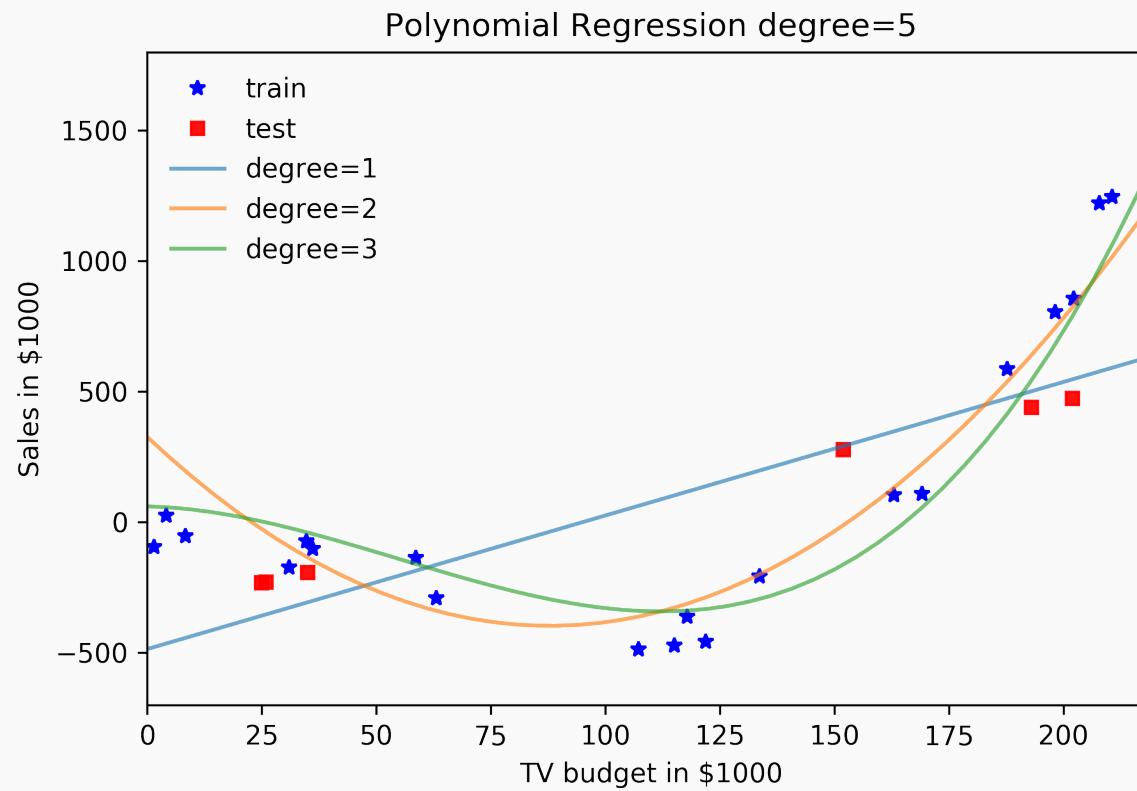
Cross Validation



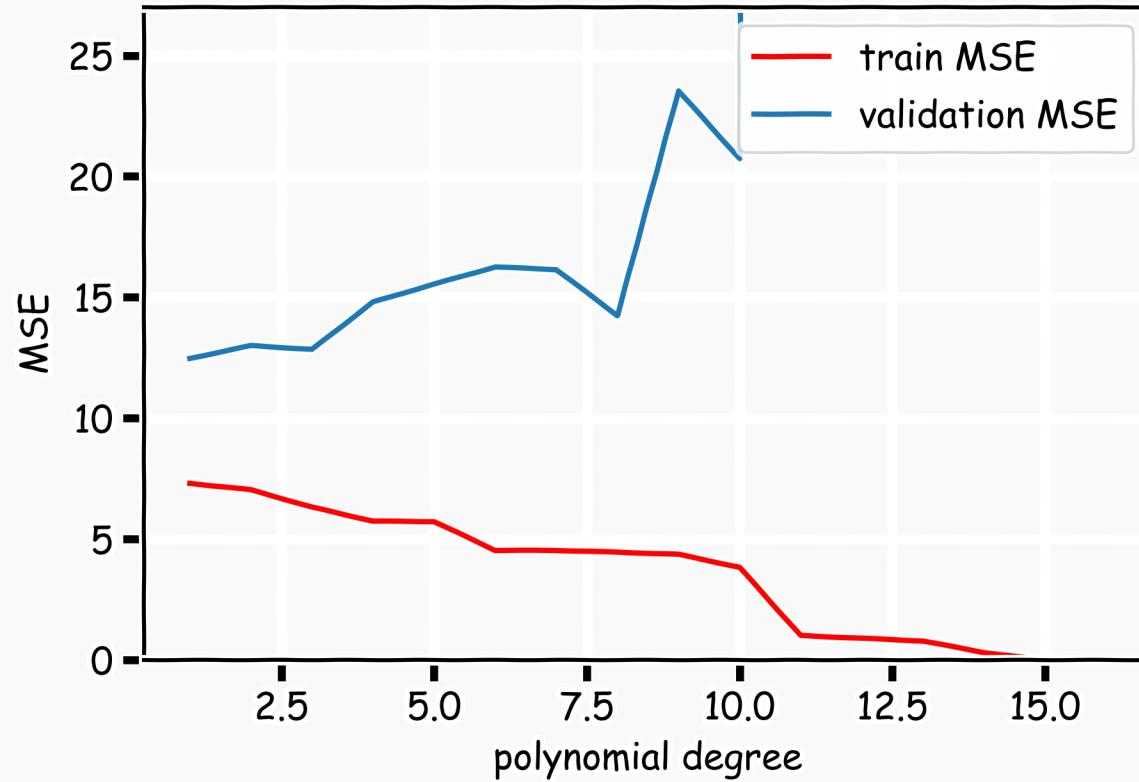
Cross Validation



Cross Validation

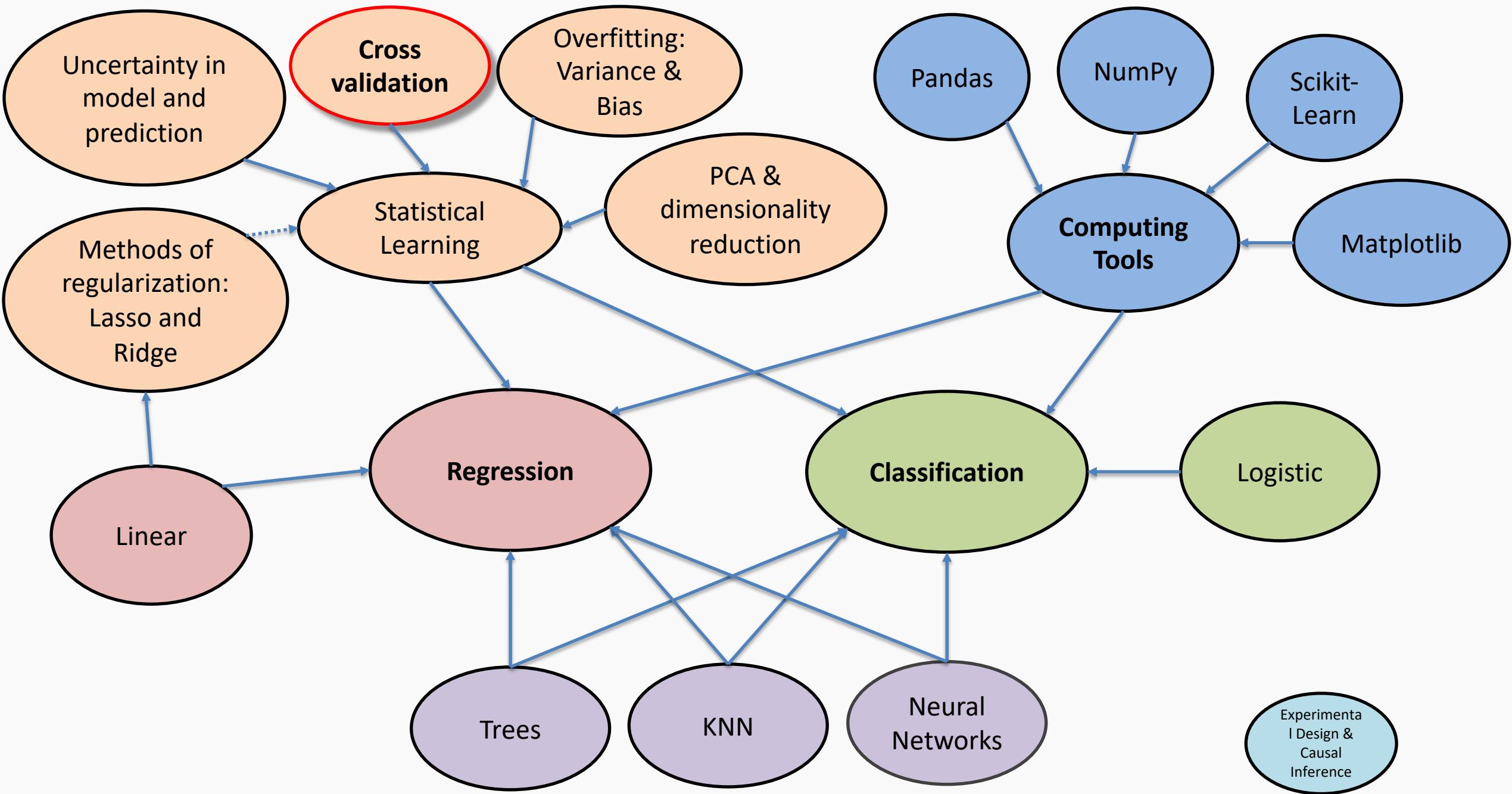


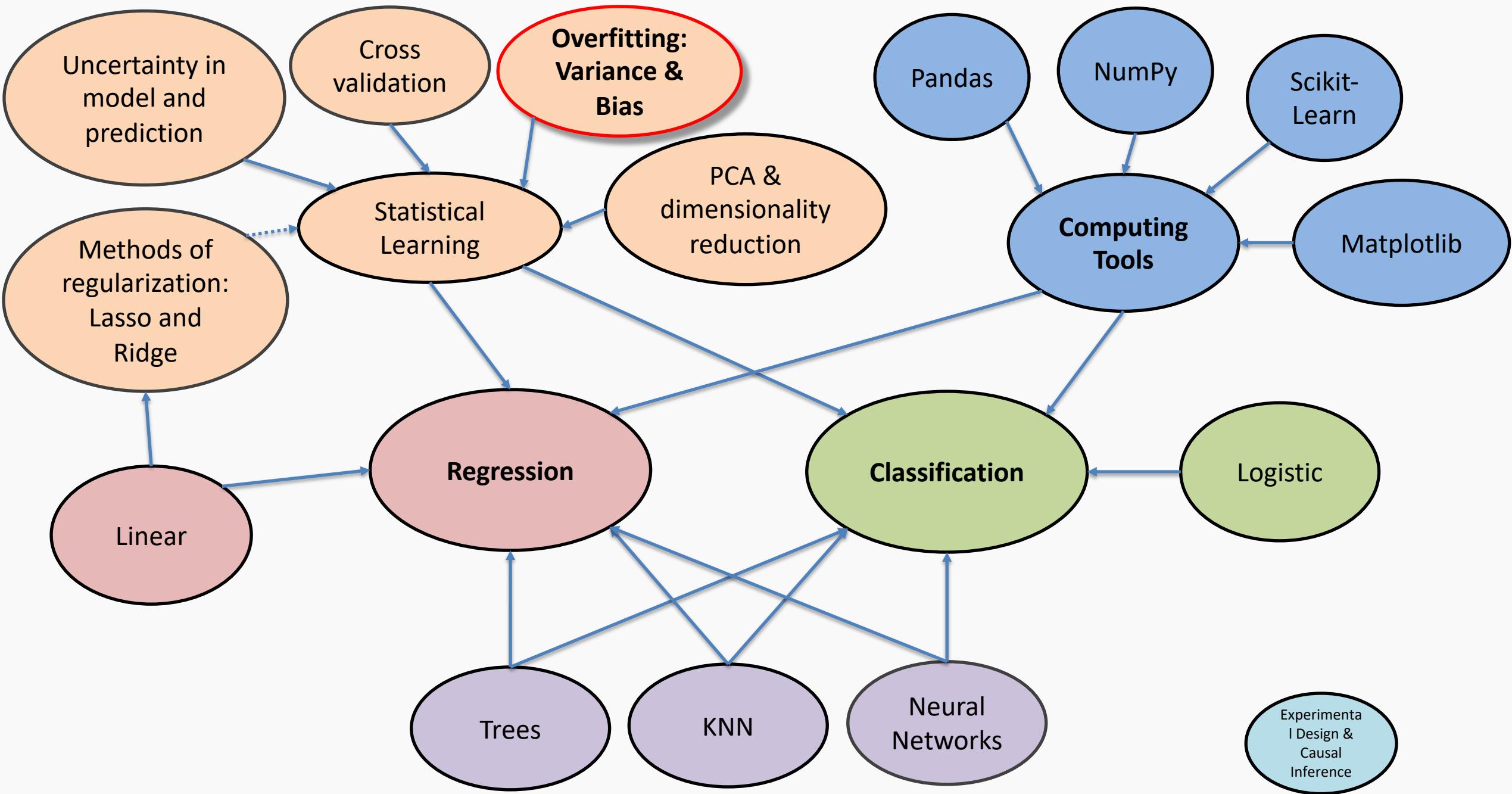
Validation

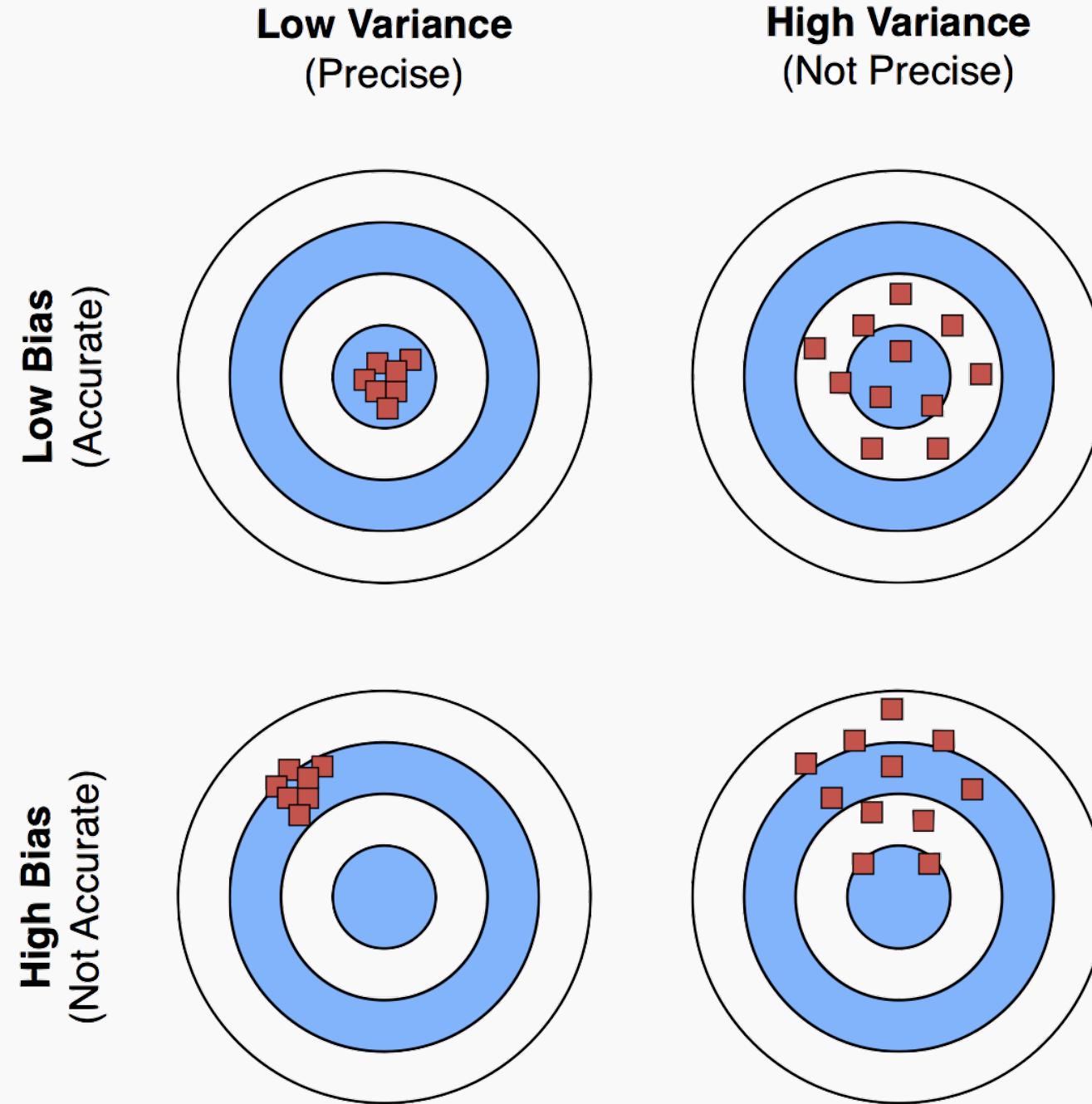


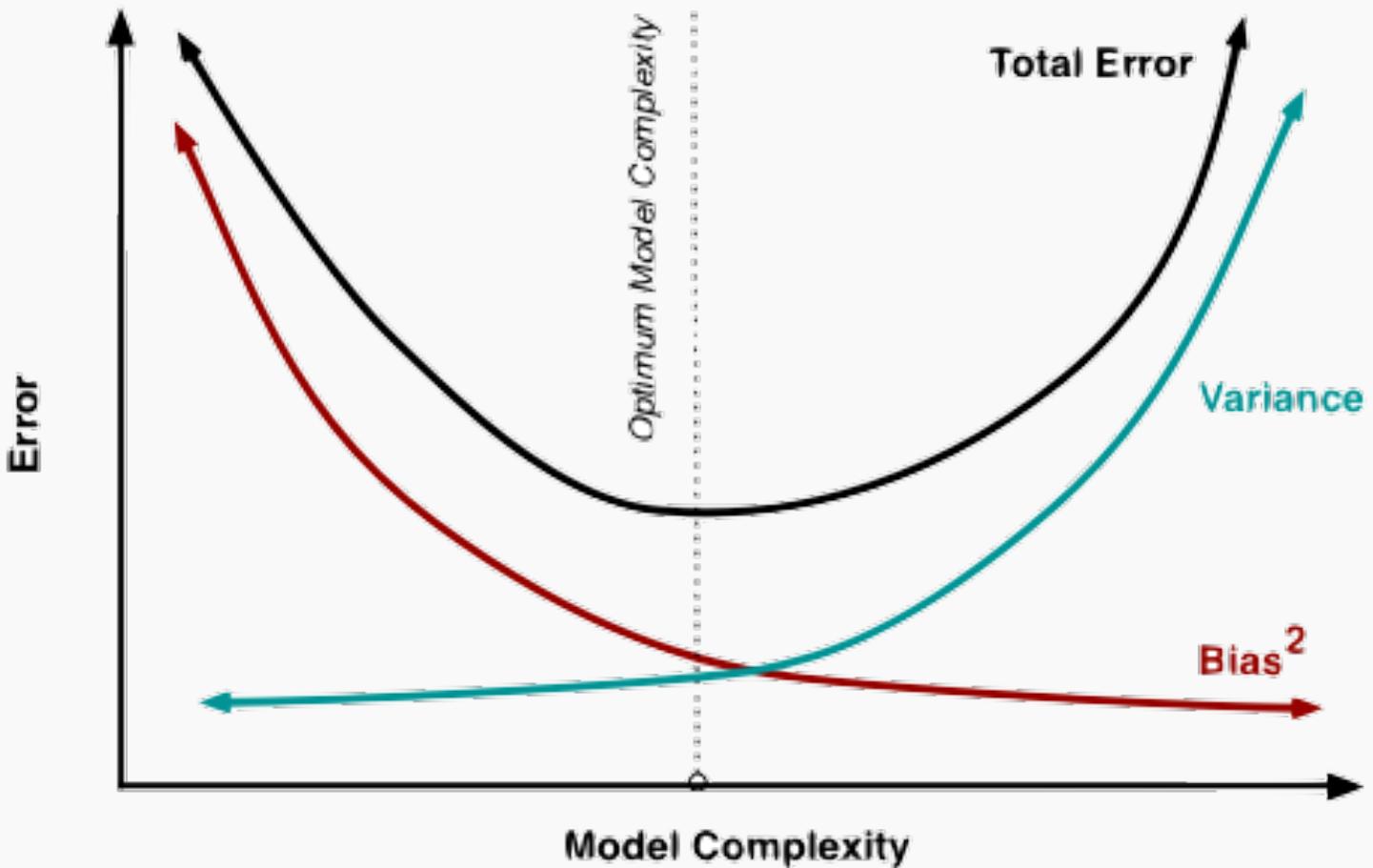
Cross Validation

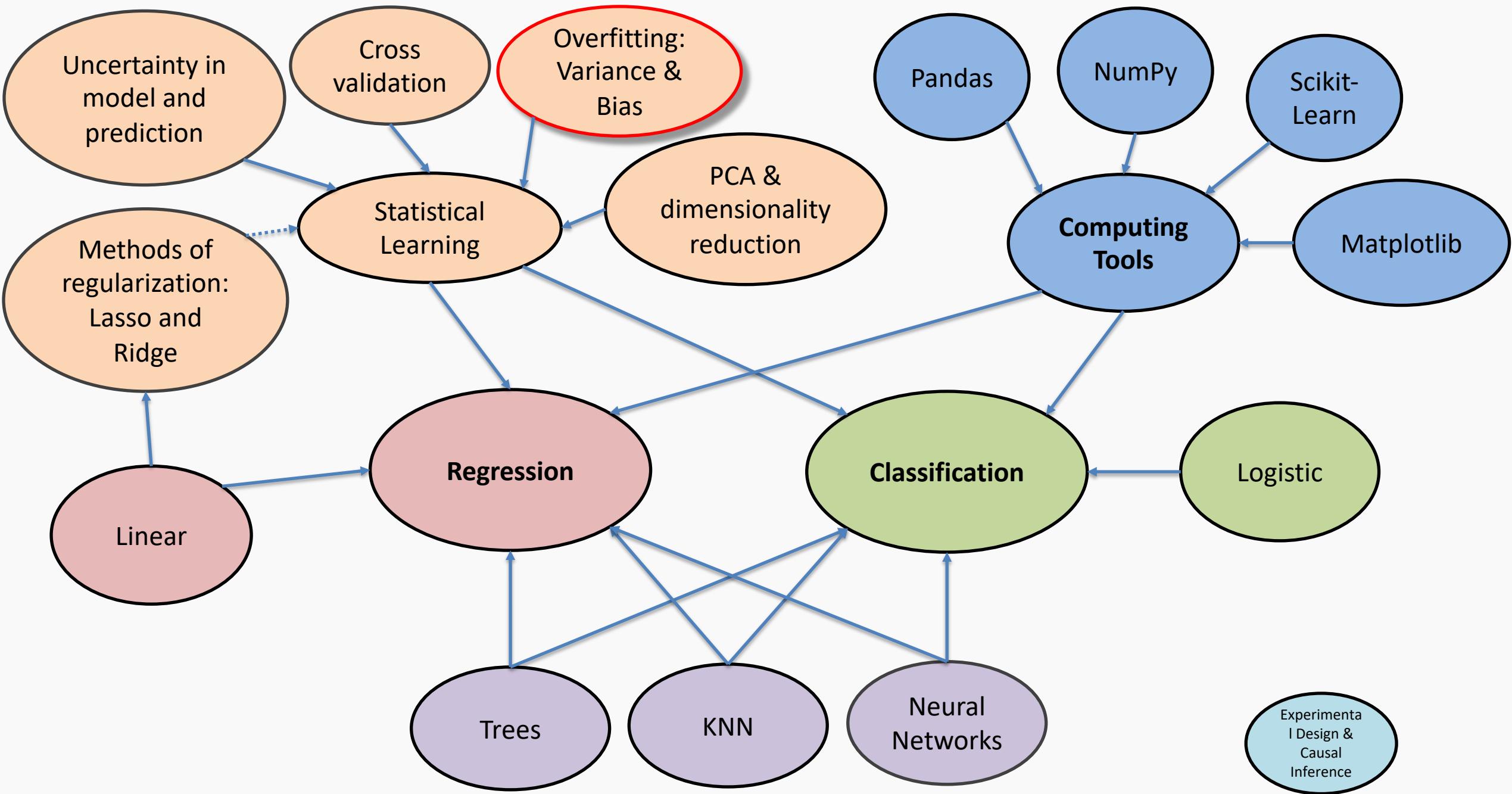


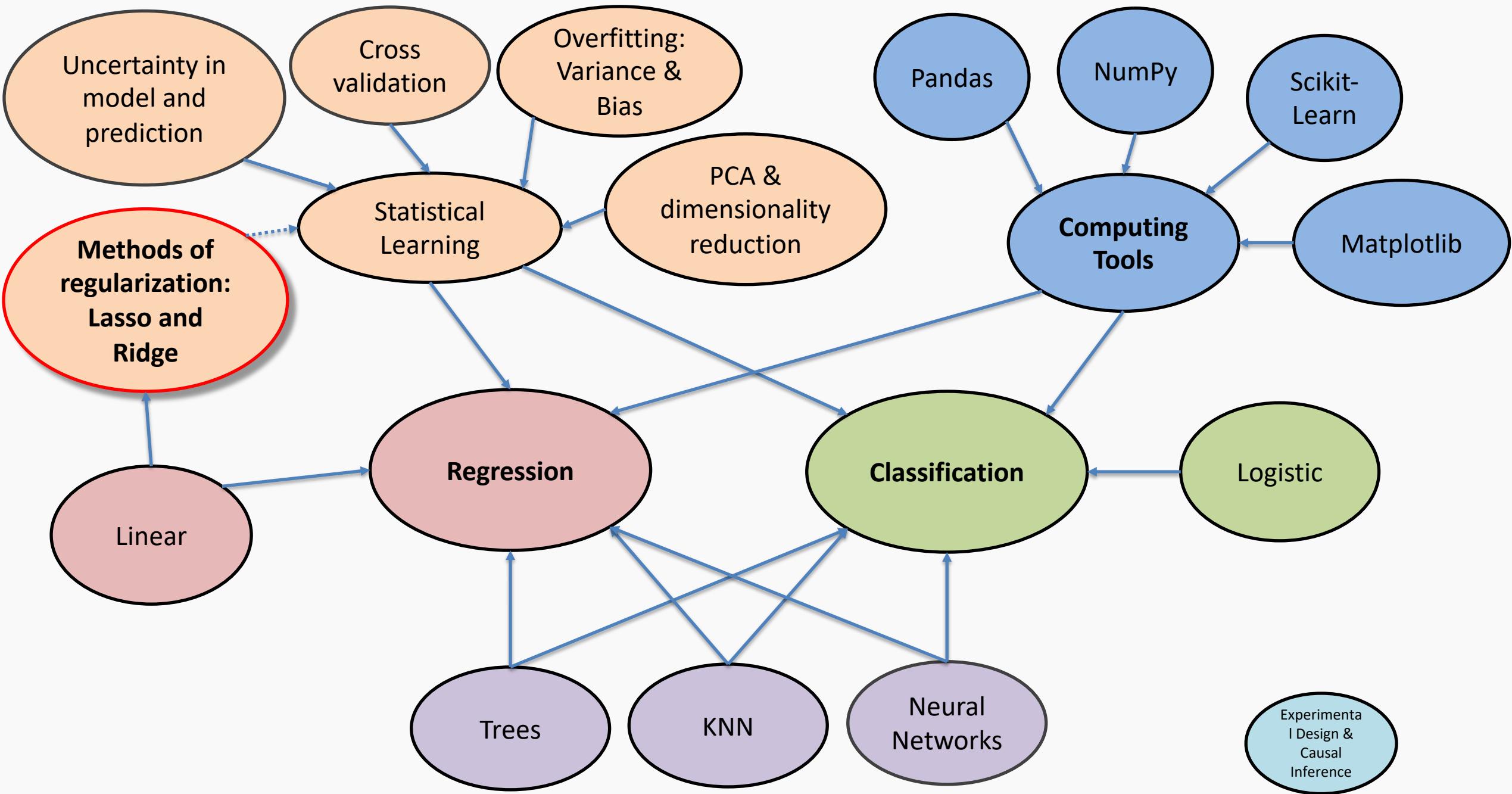




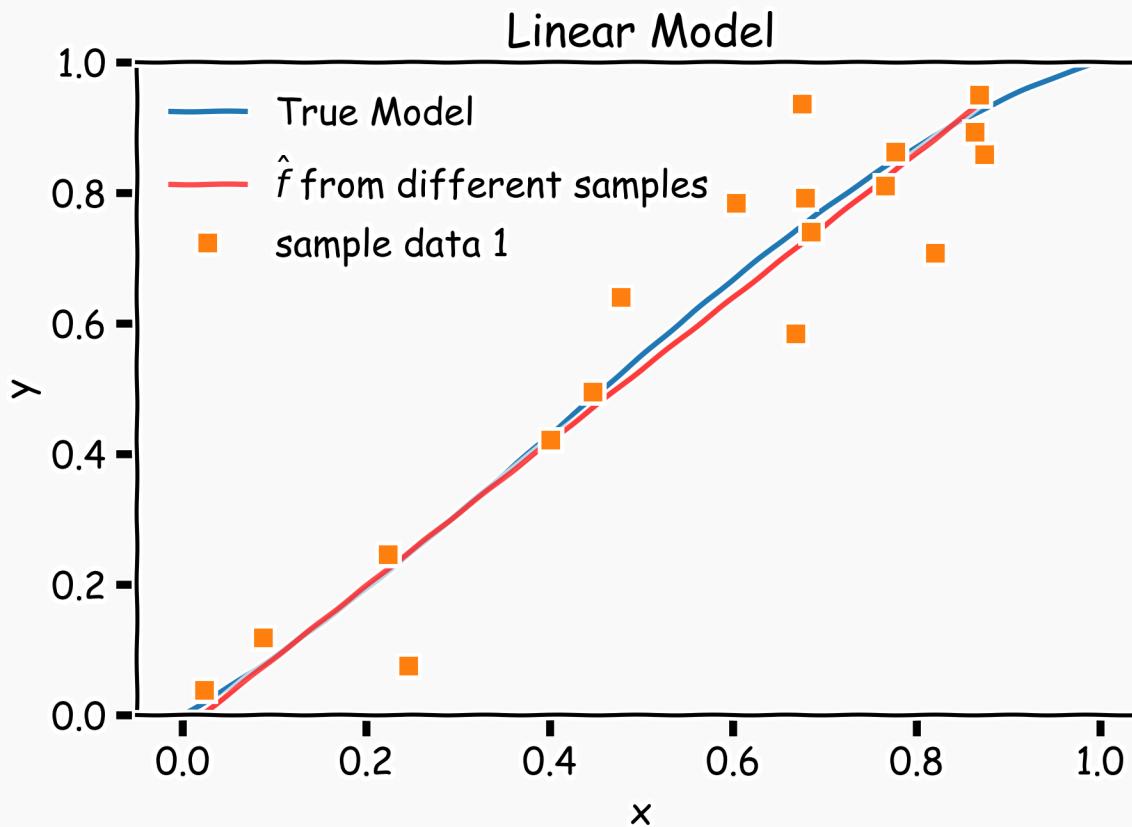




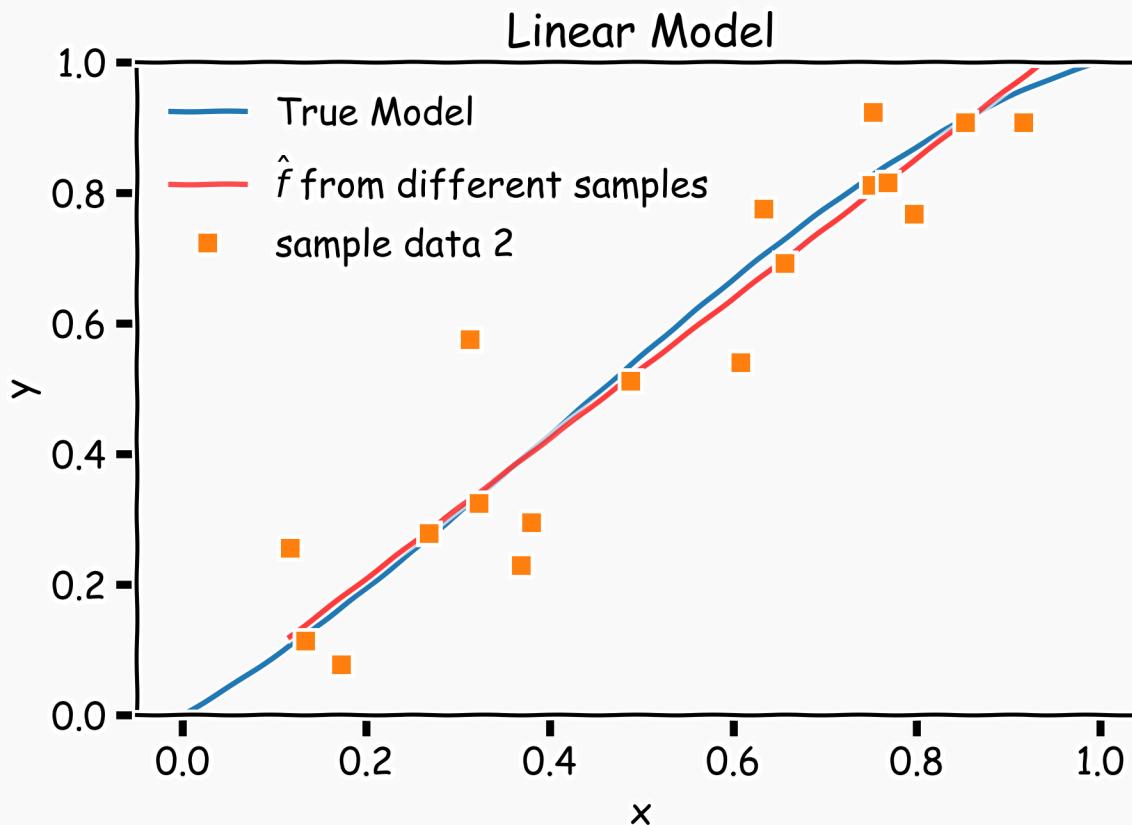




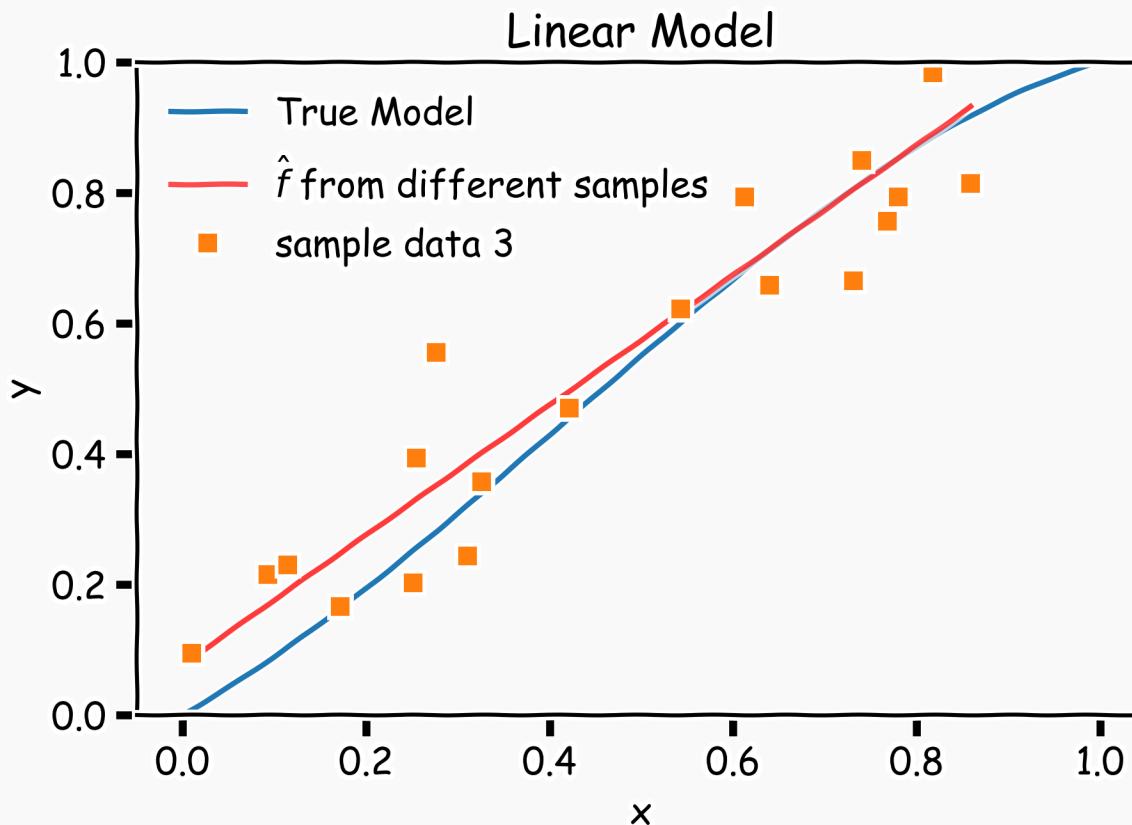
Bias vs Variance



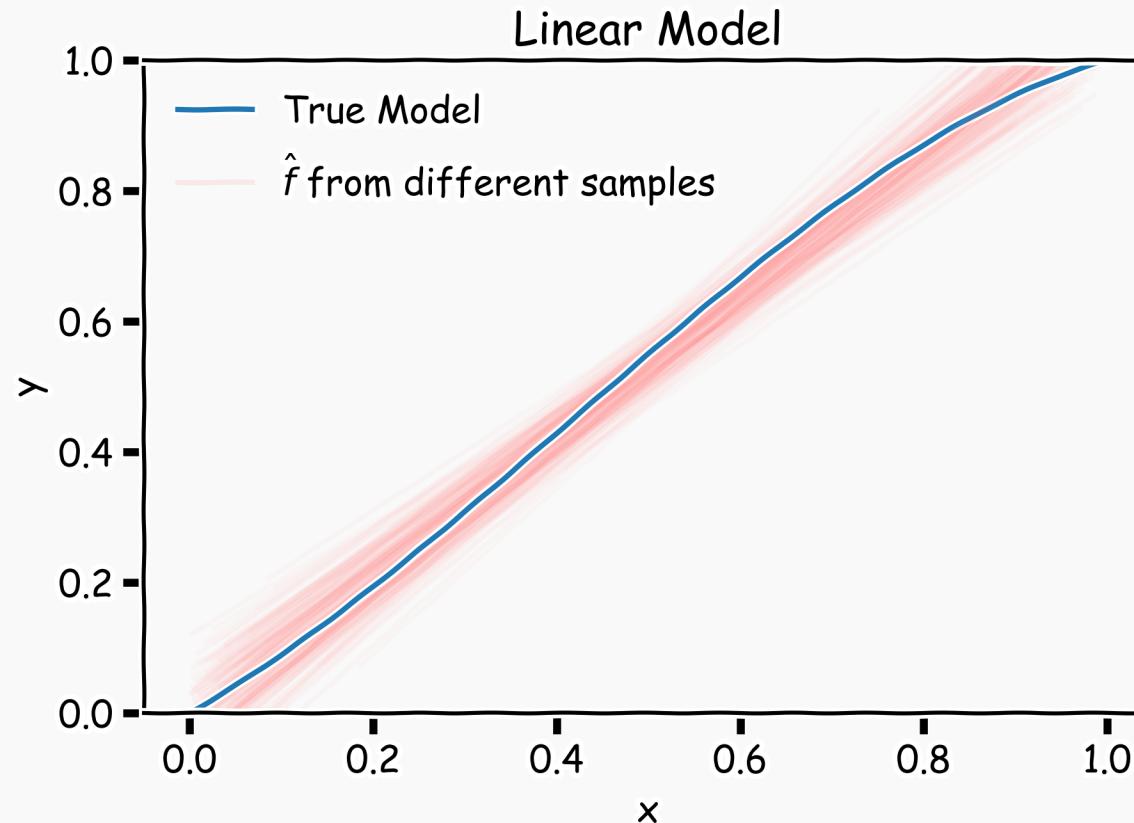
Bias vs Variance



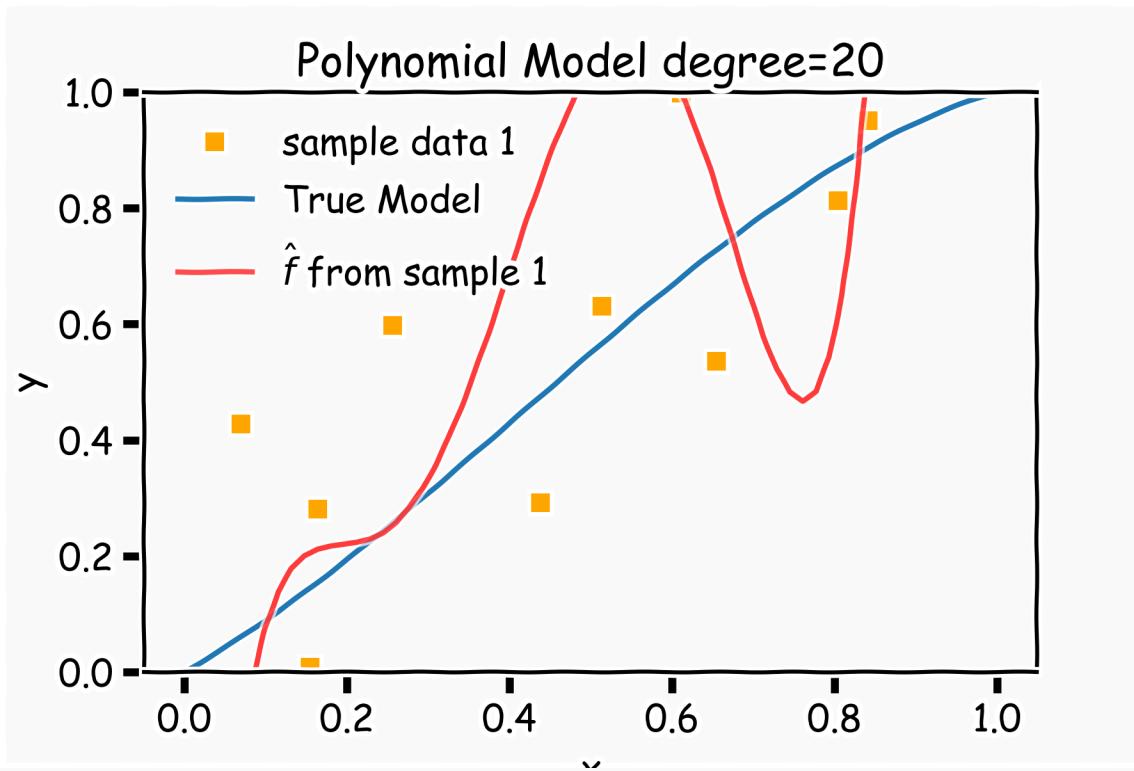
Bias vs Variance



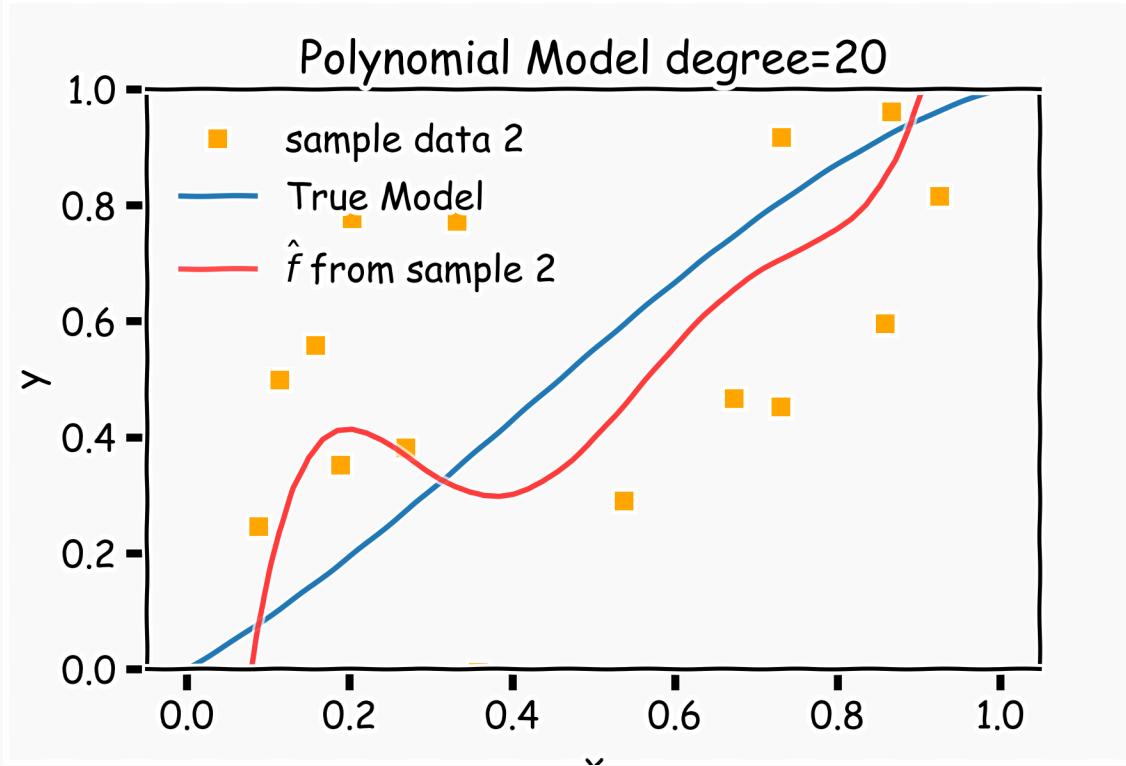
Linear models: 20 data points per line 2000 simulations



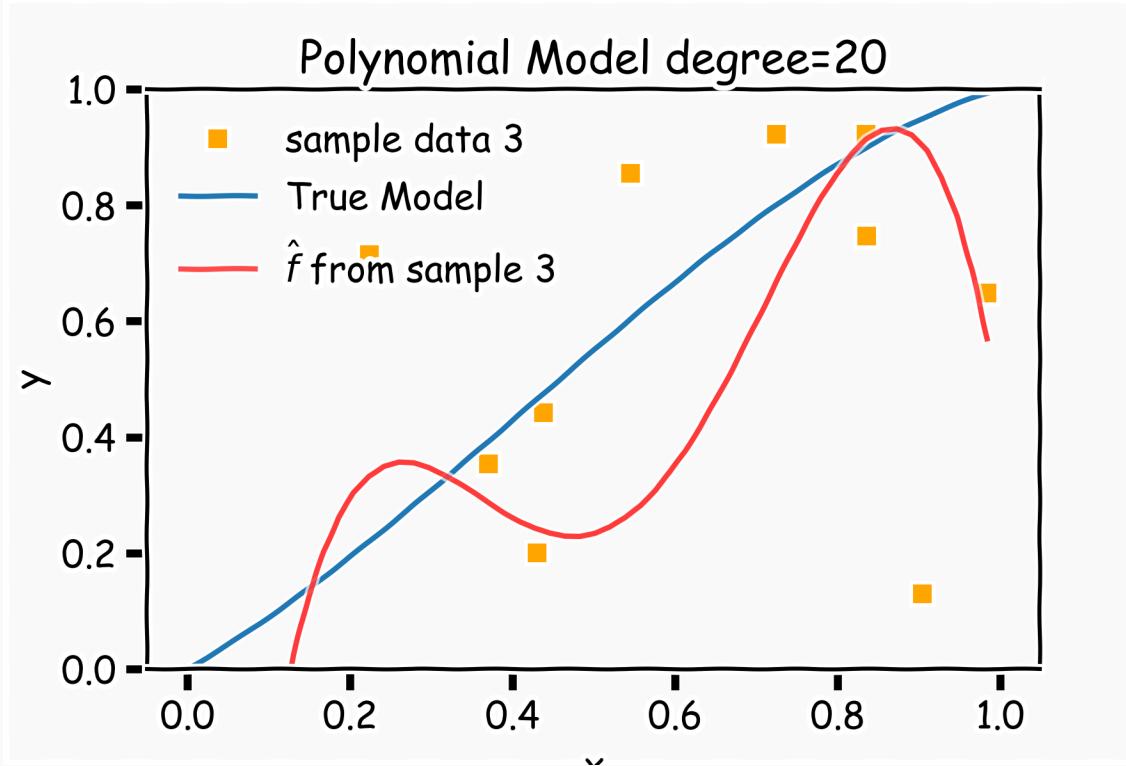
Bias vs Variance



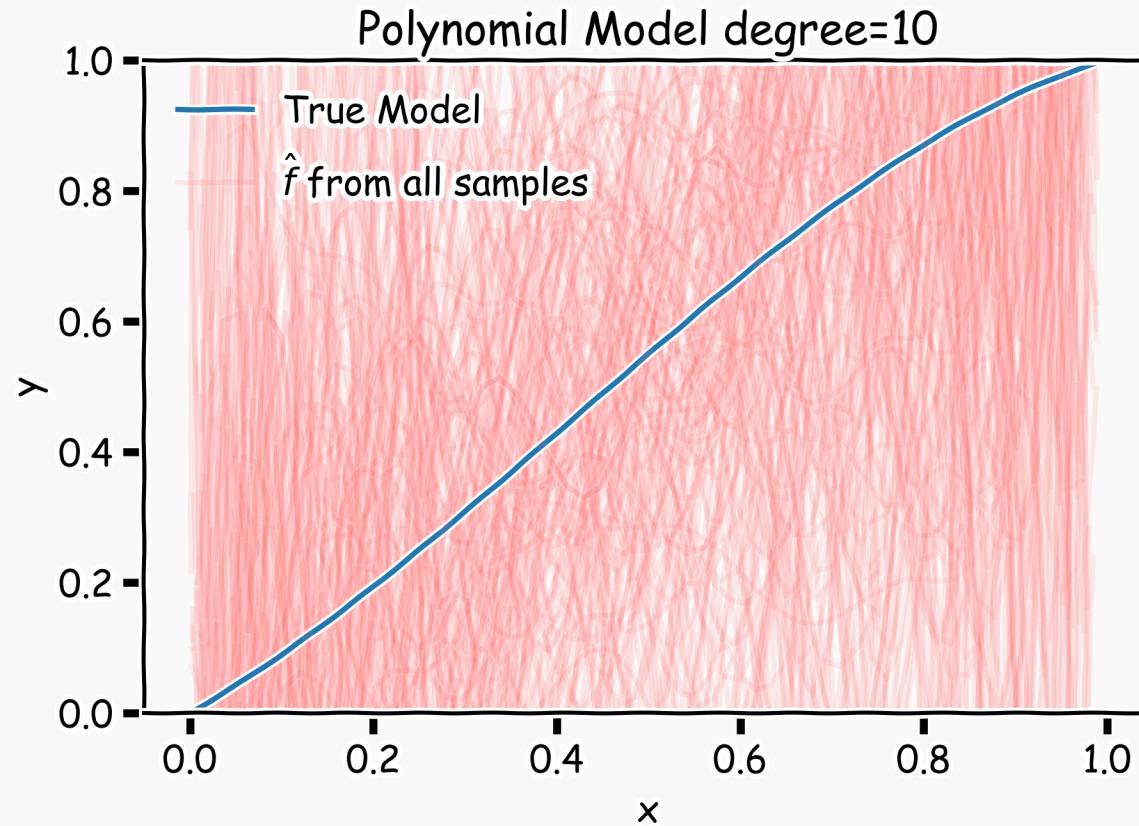
Bias vs Variance



Bias vs Variance



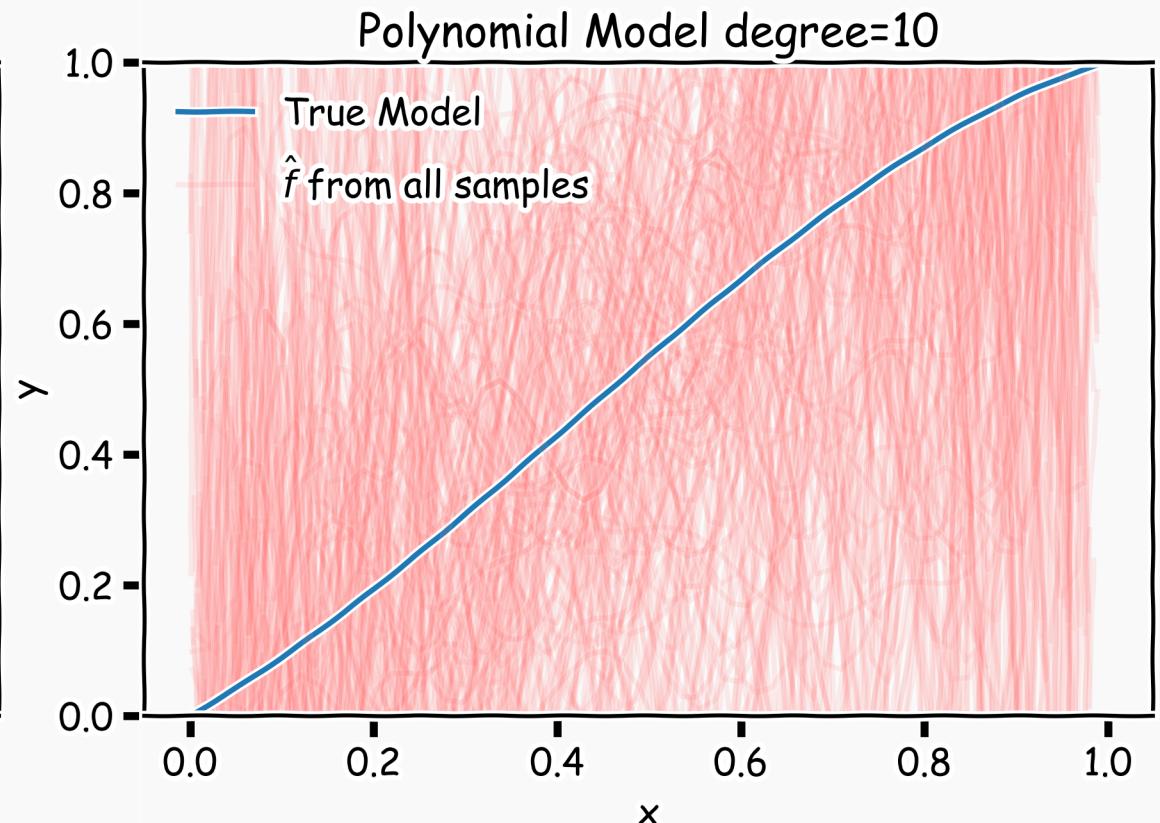
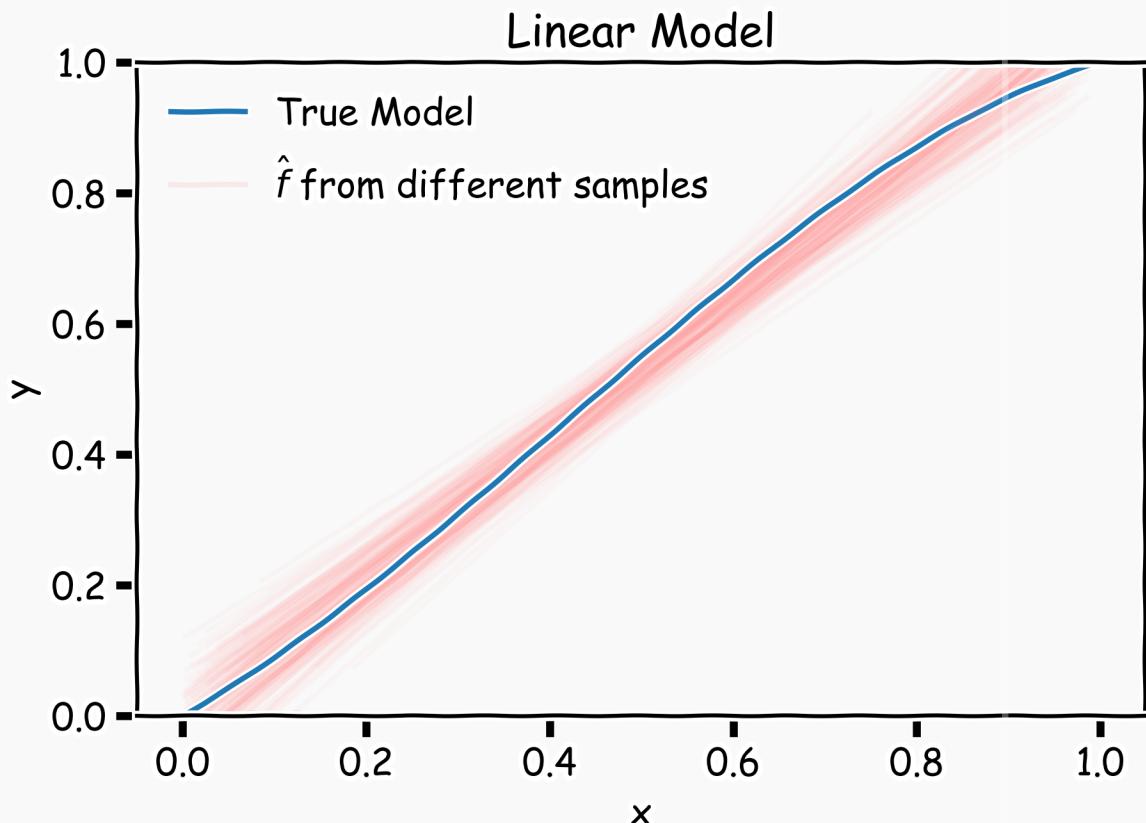
Poly 10 degree models : 20 data points per line 2000 simulations



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:

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

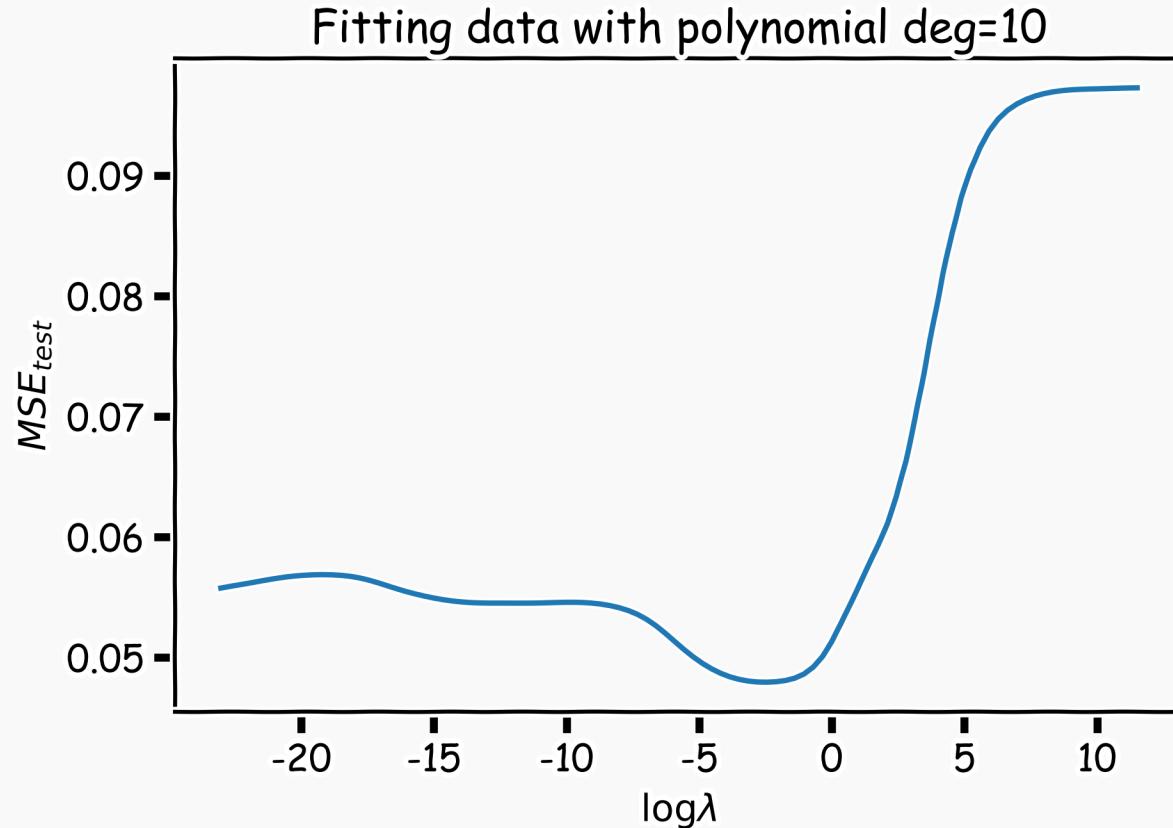
Note that $\sum_{j=1}^J \beta_j^2$ is the l_2 norm of the vector β

$$\sum_{j=1}^J \beta_j^2$$

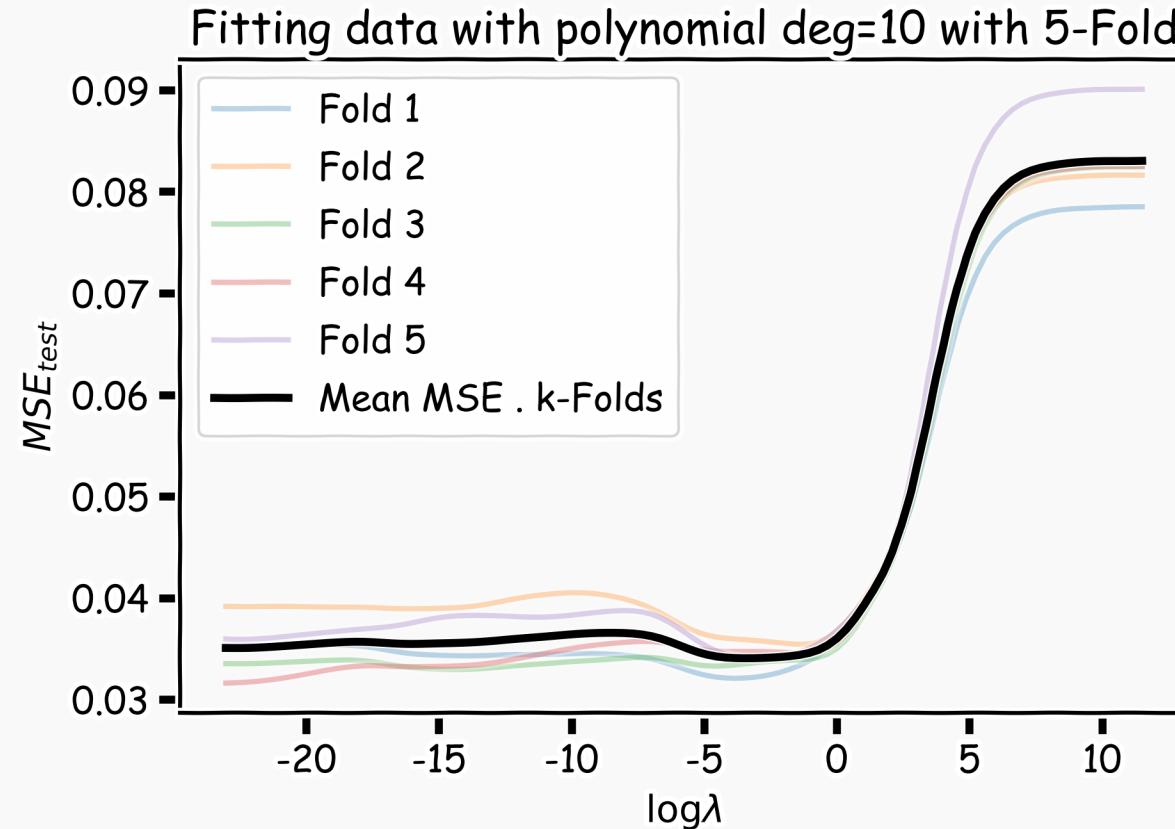
$$\sum_{j=1}^J \beta_j^2 = \|\beta\|_2^2$$

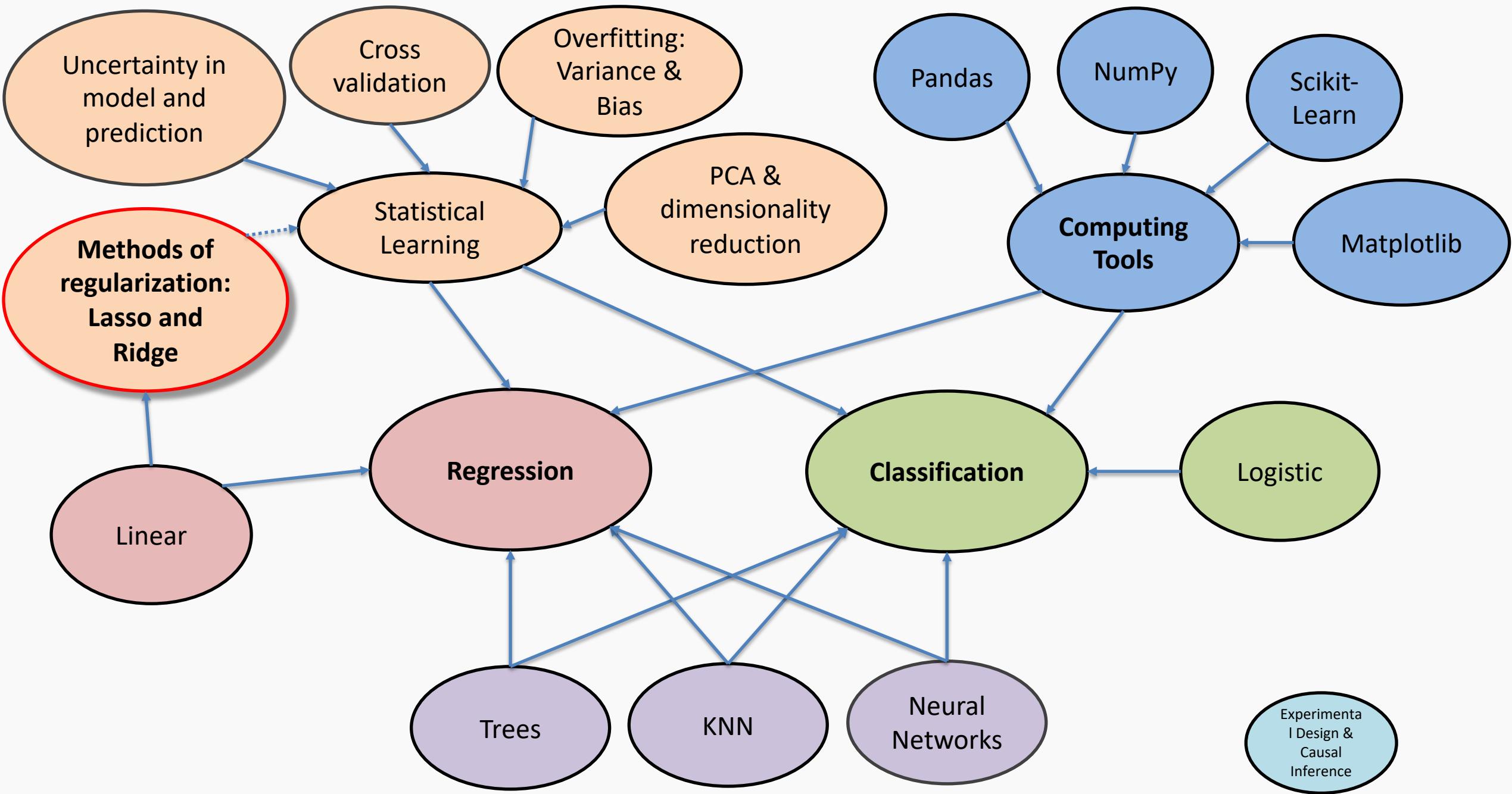


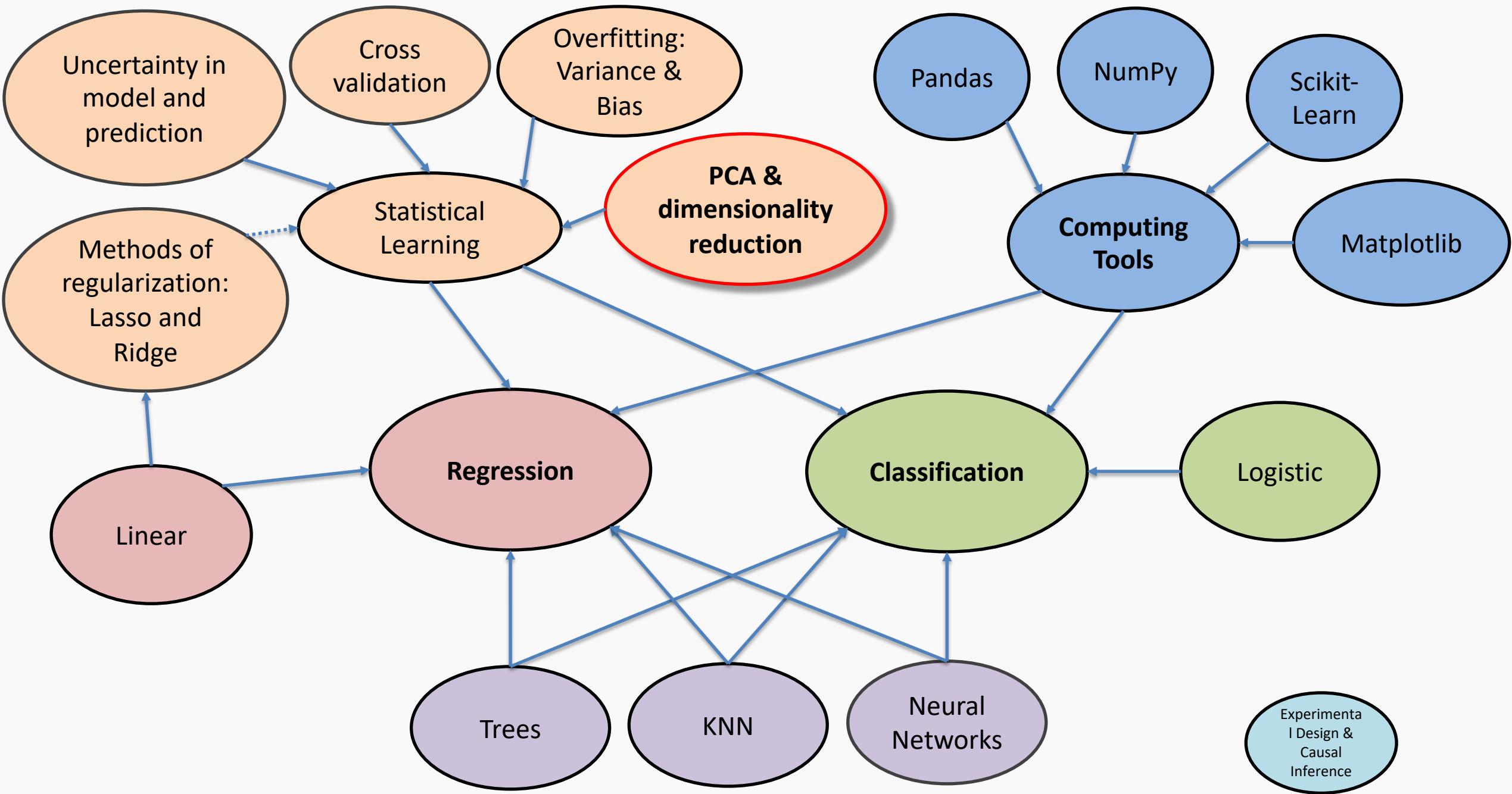
Ridge regularization with **validation** only: step by step



Ridge regularization with validation only: step by step



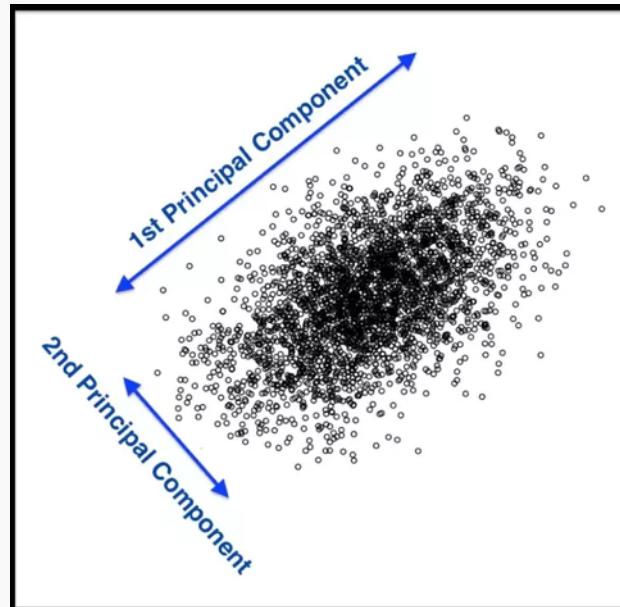




The Intuition Behind PCA

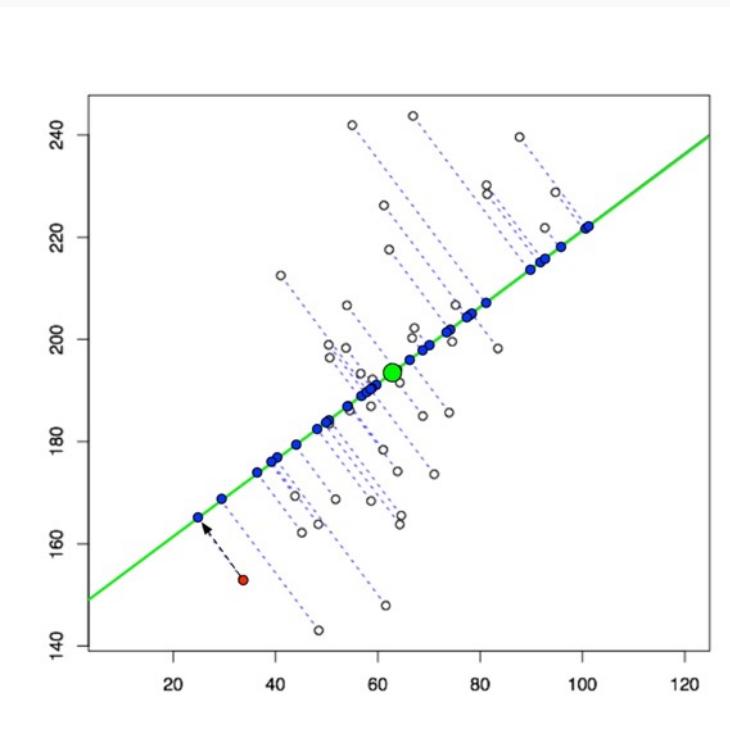
Top PCA components capture the most of amount of variation (interesting features) of the data.

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

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

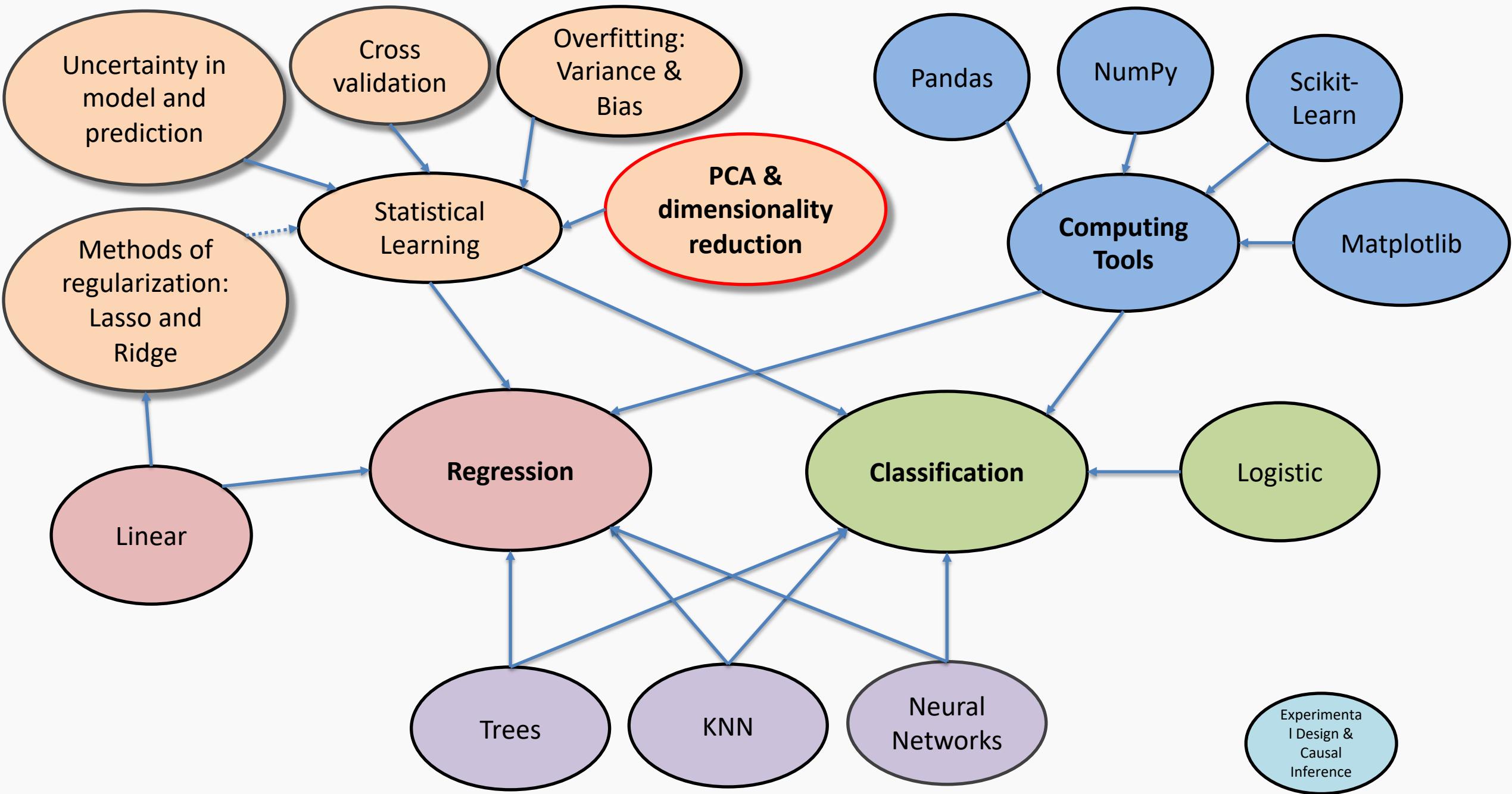
Notice that this model has a smaller number ($m+1 < p+1$) of parameters.

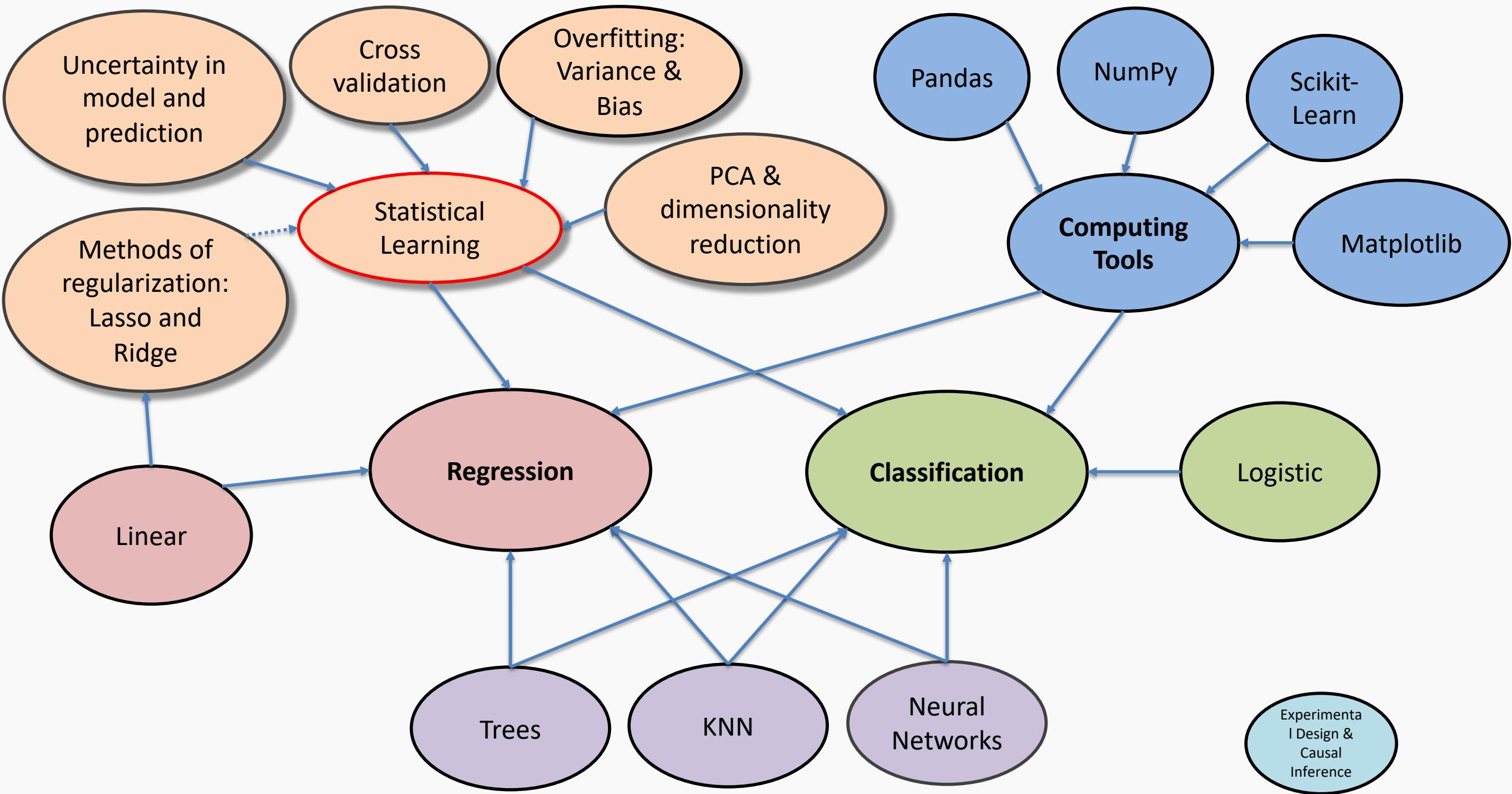
The Math behind PCA

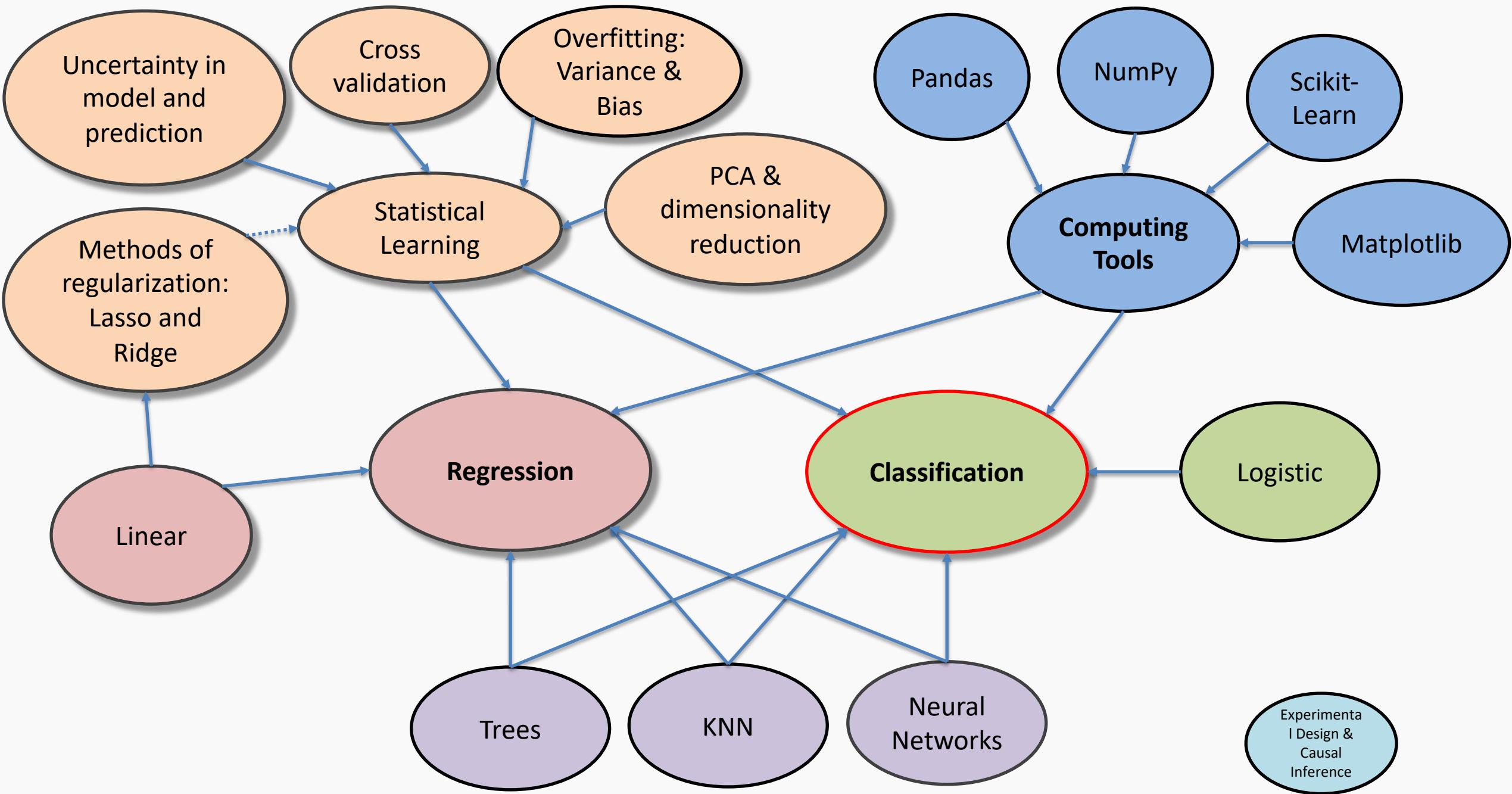
PCA is a well-known result from linear algebra. Let \mathbf{Z} be the $n \times p$ matrix consisting of columns Z_1, \dots, Z_p (the resulting PCA vectors), \mathbf{X} be the $n \times p$ matrix of X_1, \dots, X_p of the original data variables (each standardized to have mean zero and variance one, and 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:

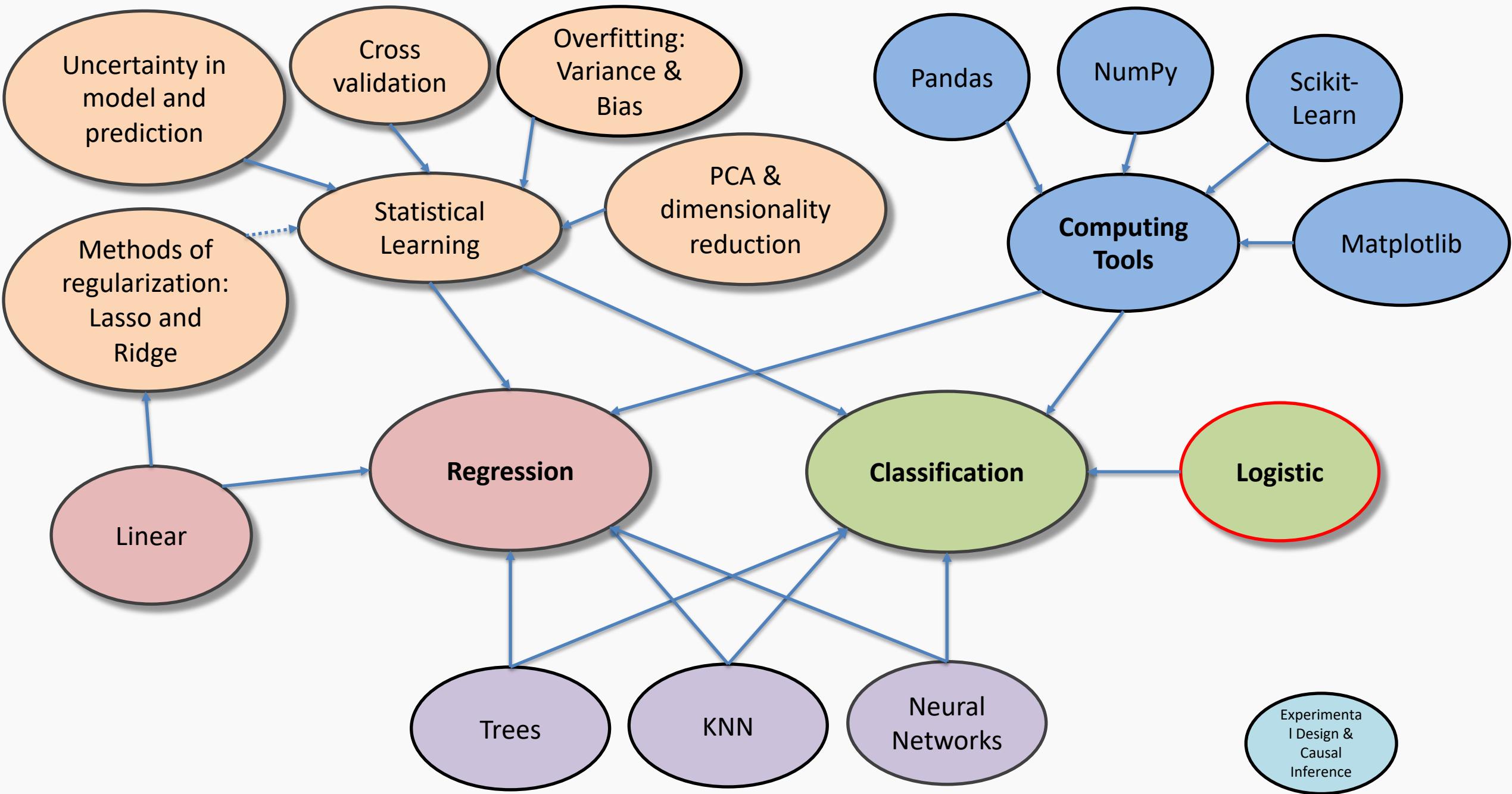
$$\mathbf{Z}_{n \times p} = \mathbf{X}_{n \times p} \mathbf{W}_{p \times p}$$





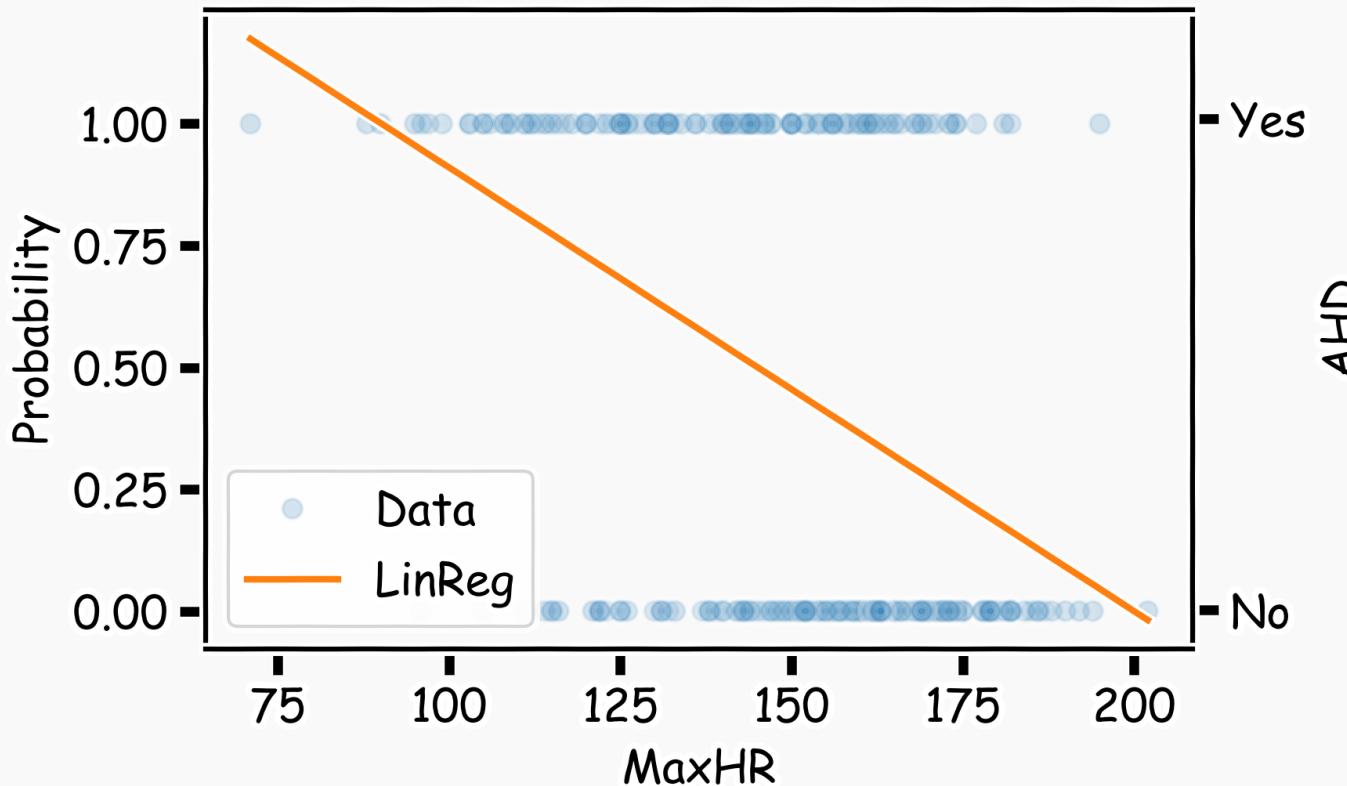






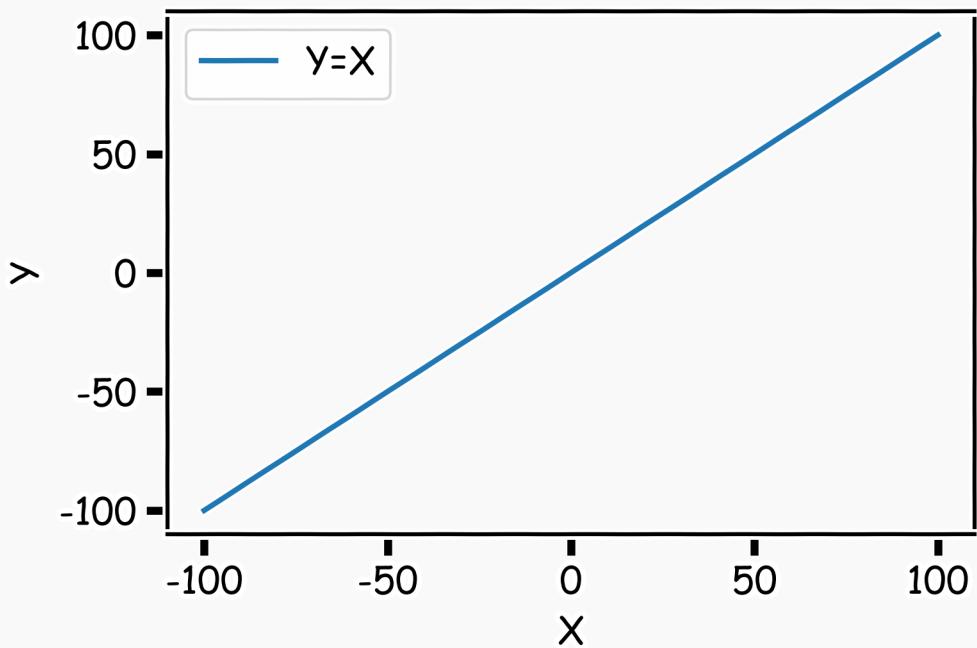
Even Simpler Classification Problem: Binary Response (cont)

What could go wrong with this linear regression model?



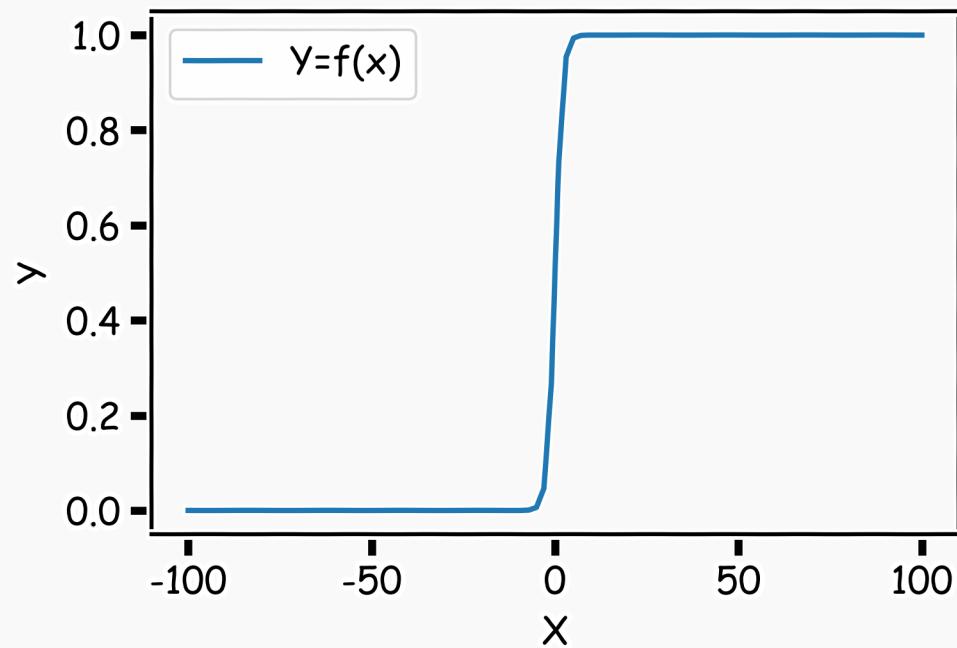
Pavlos Game #45

Think of a function that would do this for us

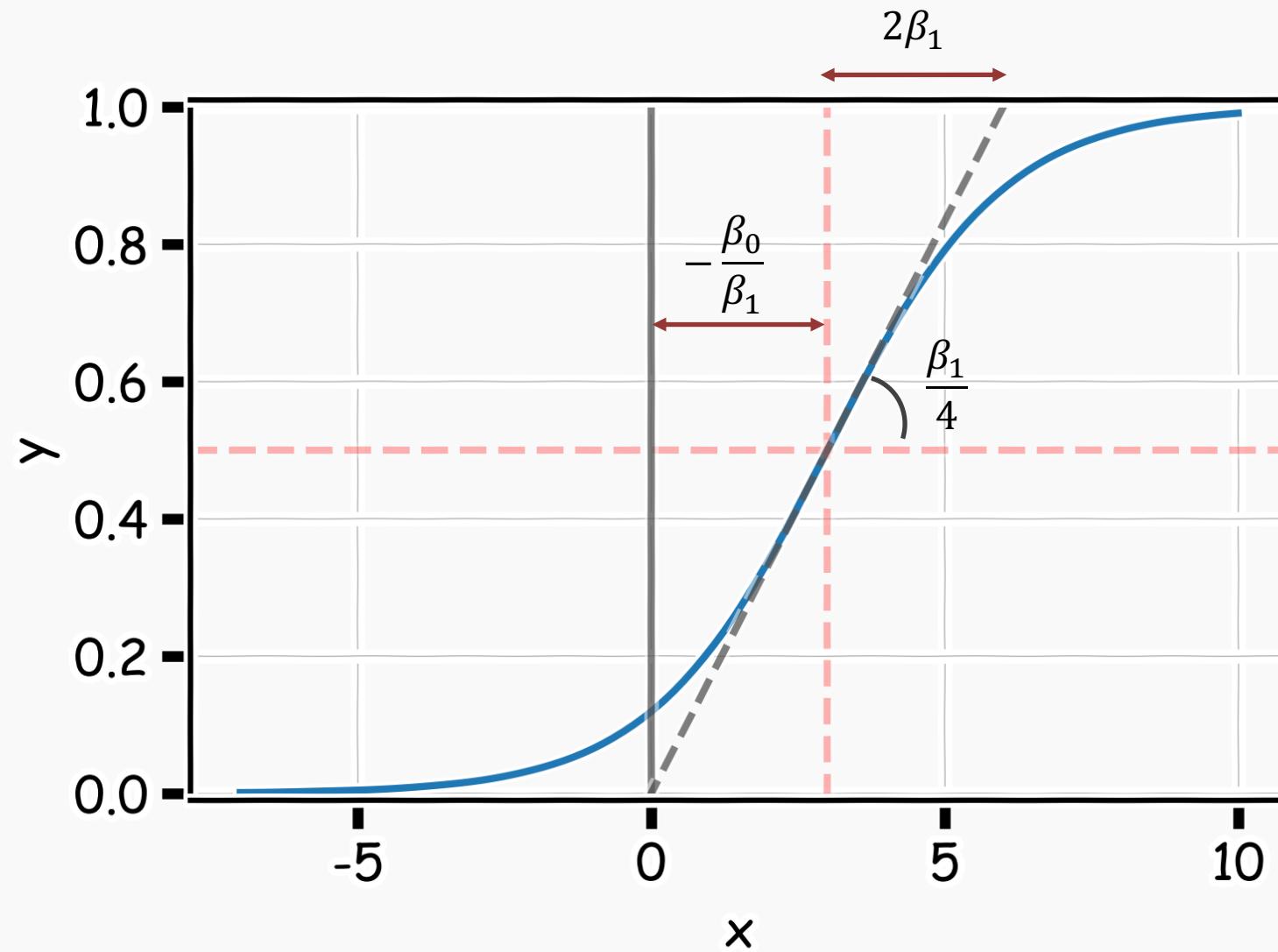


$$Y = f(x)$$

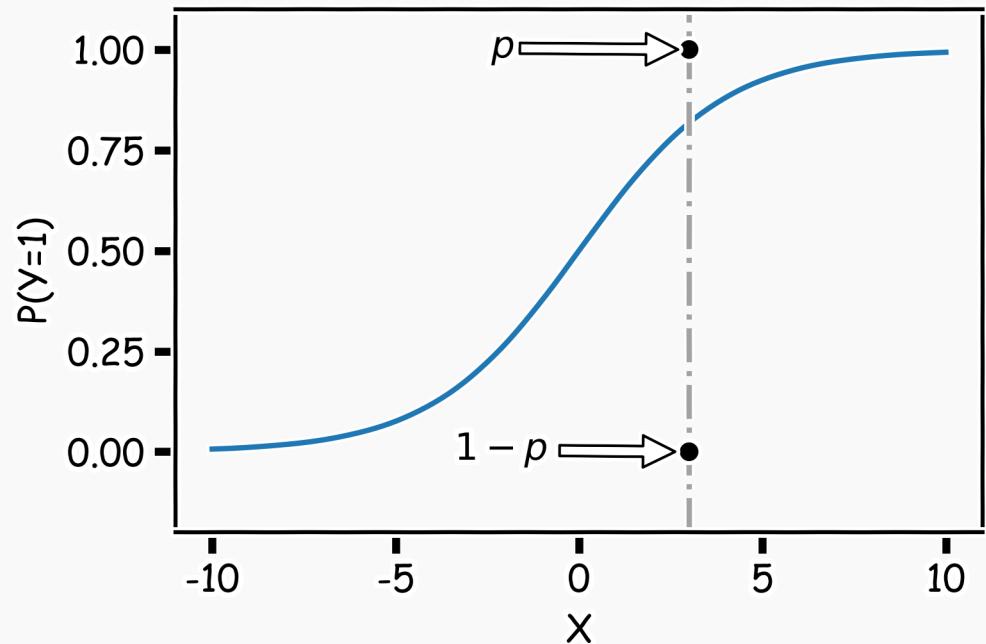
A large, solid grey arrow points from the right side of the first graph towards the second graph, indicating a transformation or mapping.



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?

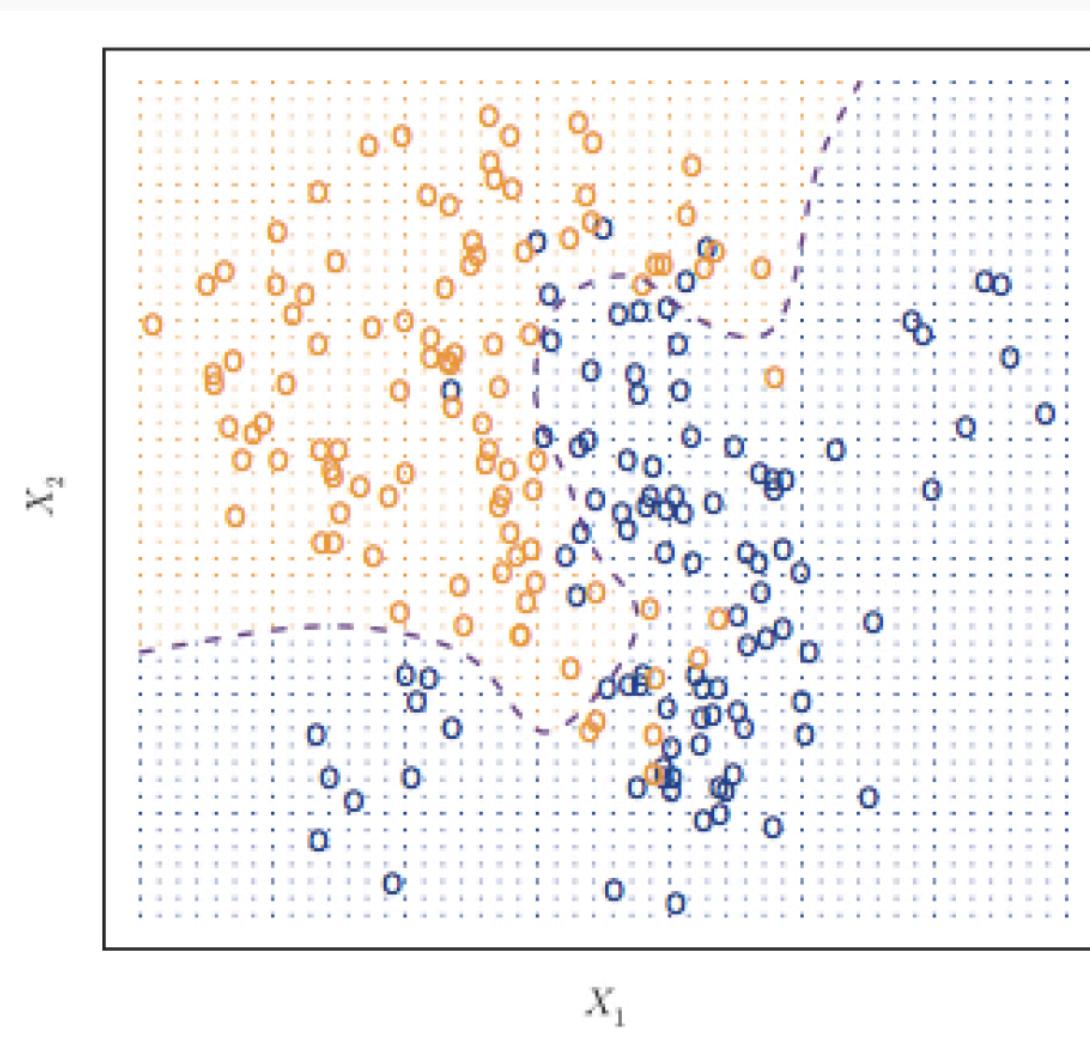
Differentiate, equate to zero and solve for it!

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

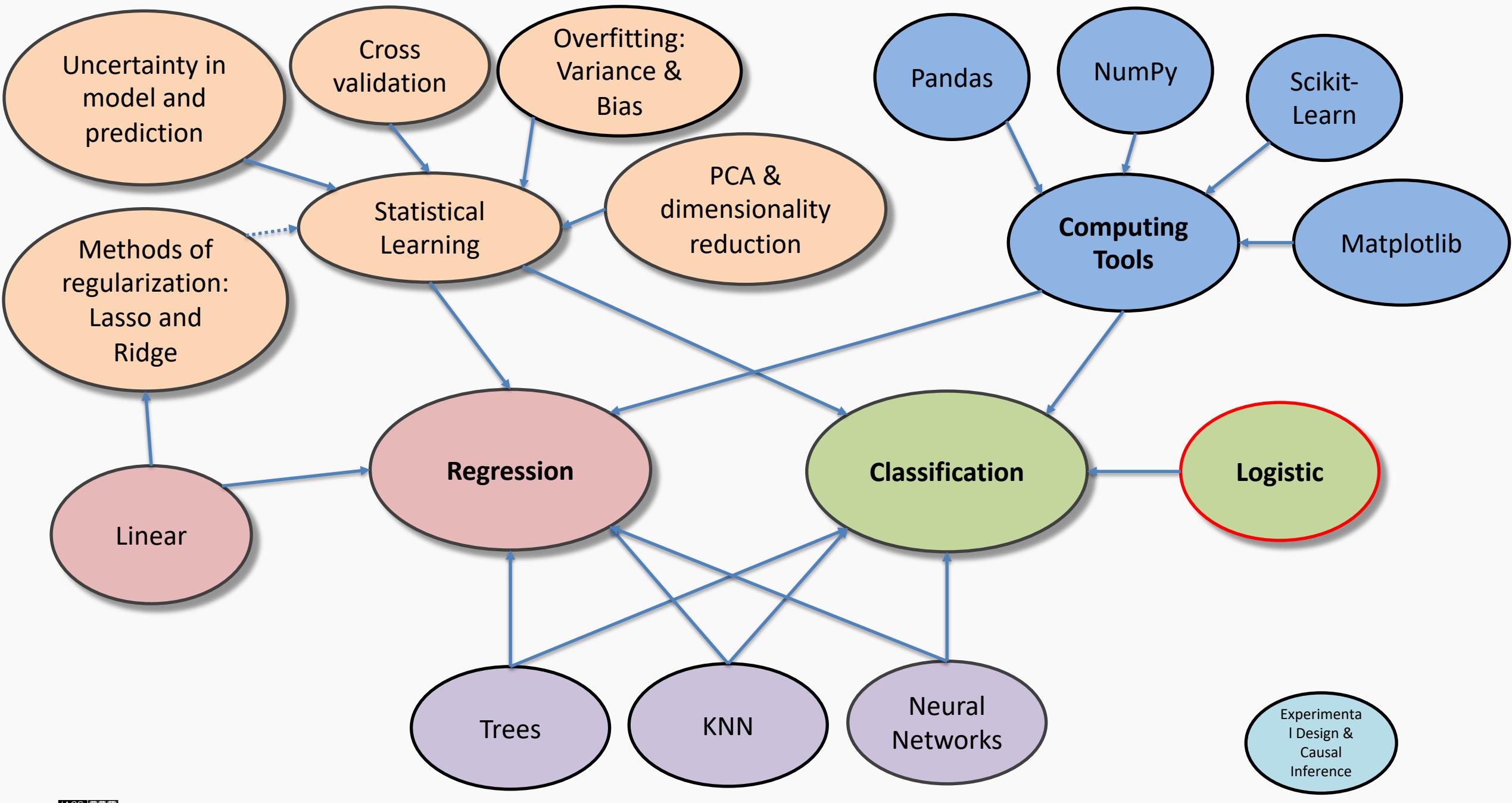
Earlier we saw the general form of *simple* logistic regression, meaning when there is just one predictor used in the model. What was the model statement (in terms of linear predictors)?

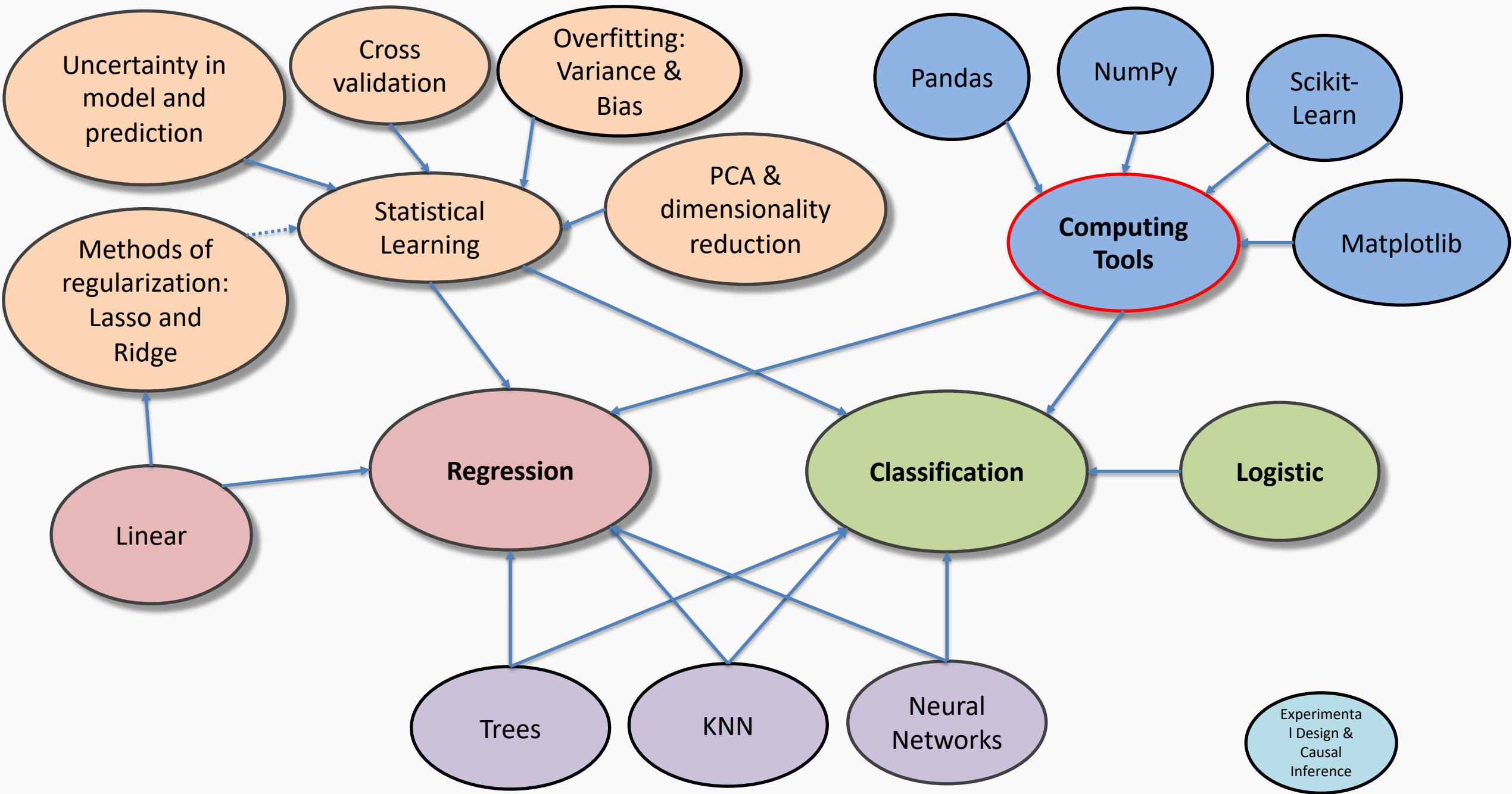
$$\log \left(\frac{P(Y = 1)}{1 - P(Y = 1)} \right) = \beta_0 + \beta_1 X$$

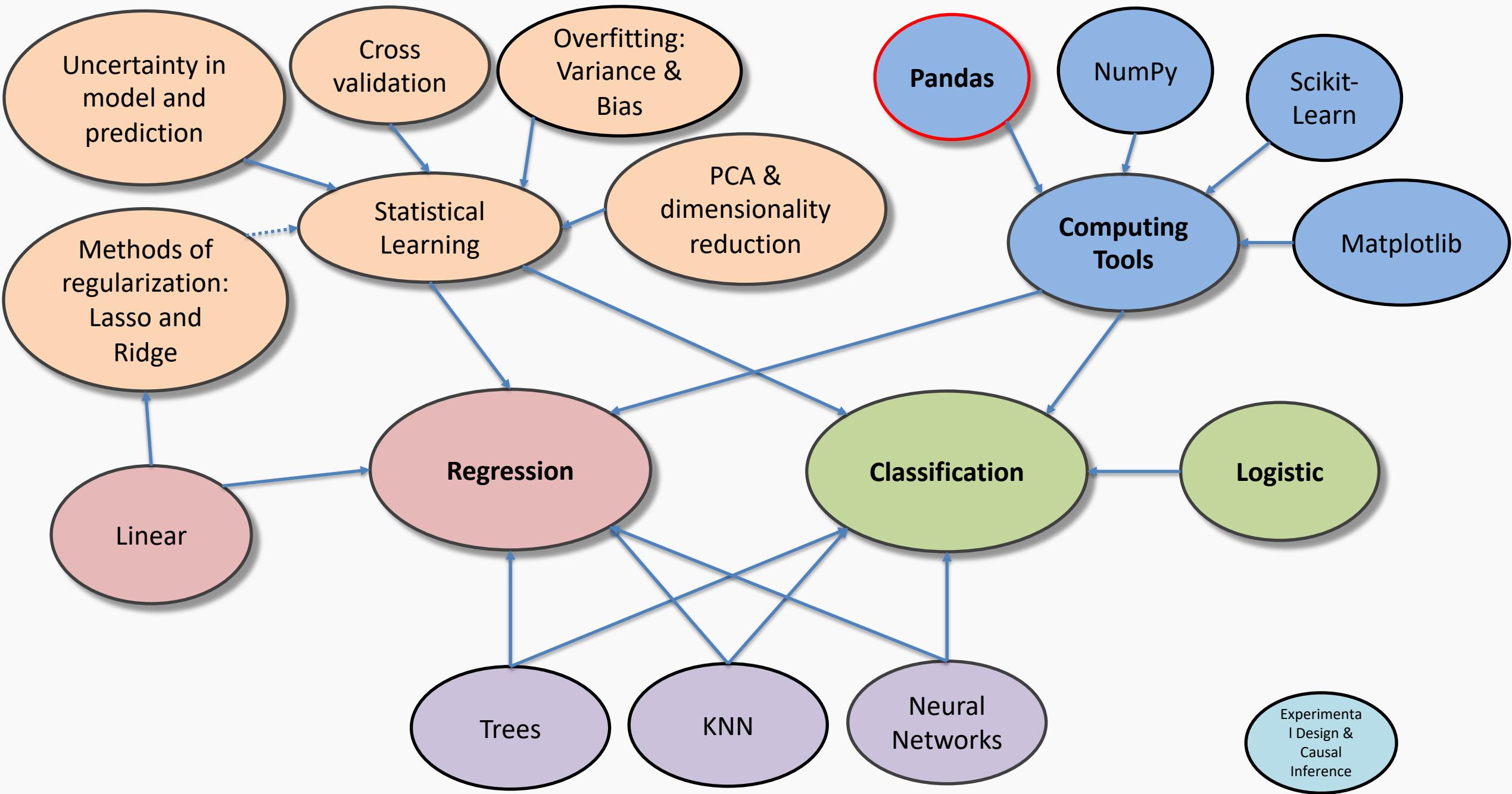
Multiple logistic regression is a generalization to multiple predictors. More specifically we can define a multiple logistic regression model to predict $P(Y = 1)$ as such:

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









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

```
>>> 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

```
>>> 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']>1200000000]
```

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

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
Cumulative 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:

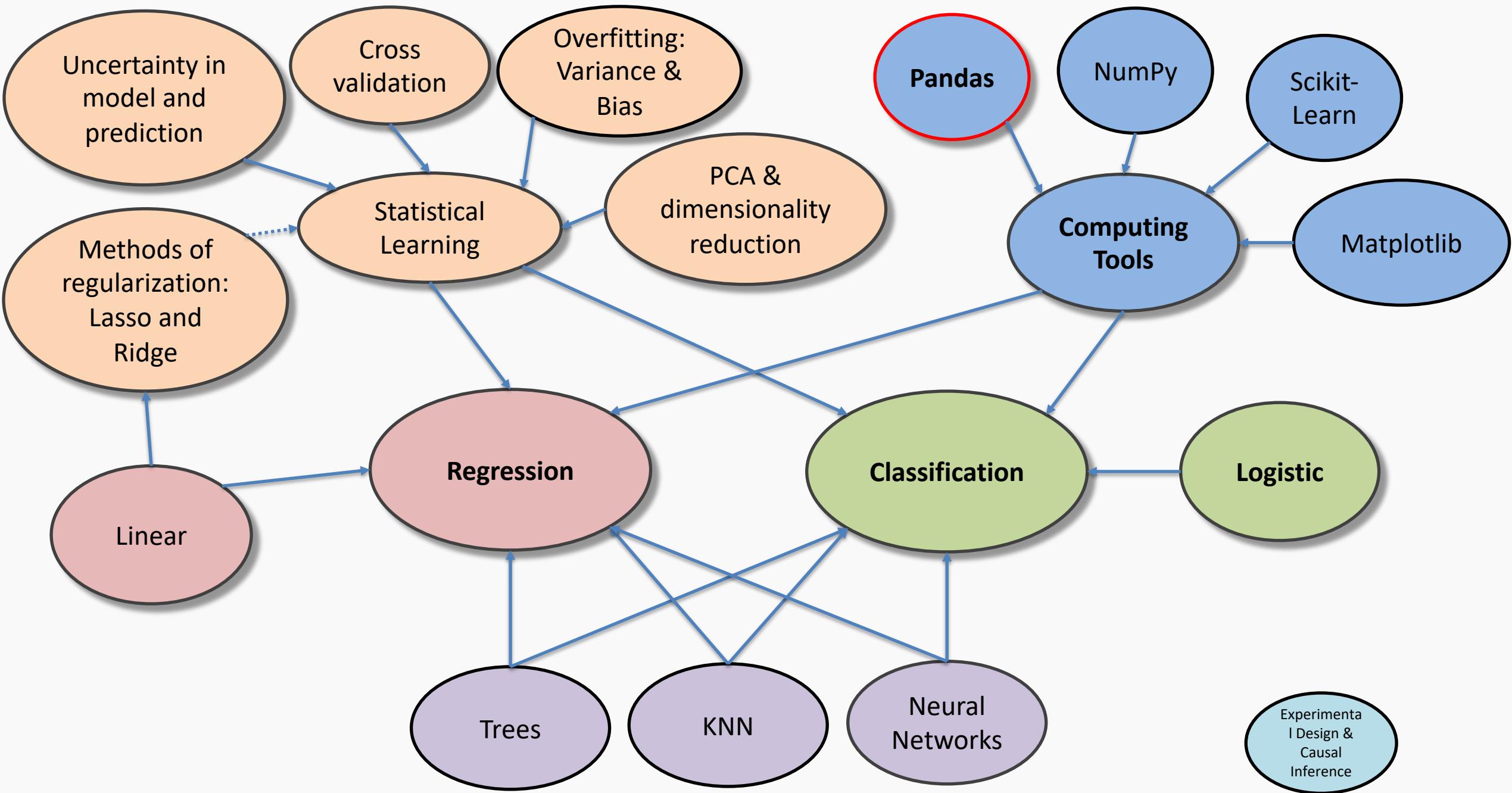
```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a    10.0
b    NaN
c     5.0
d     7.0
```

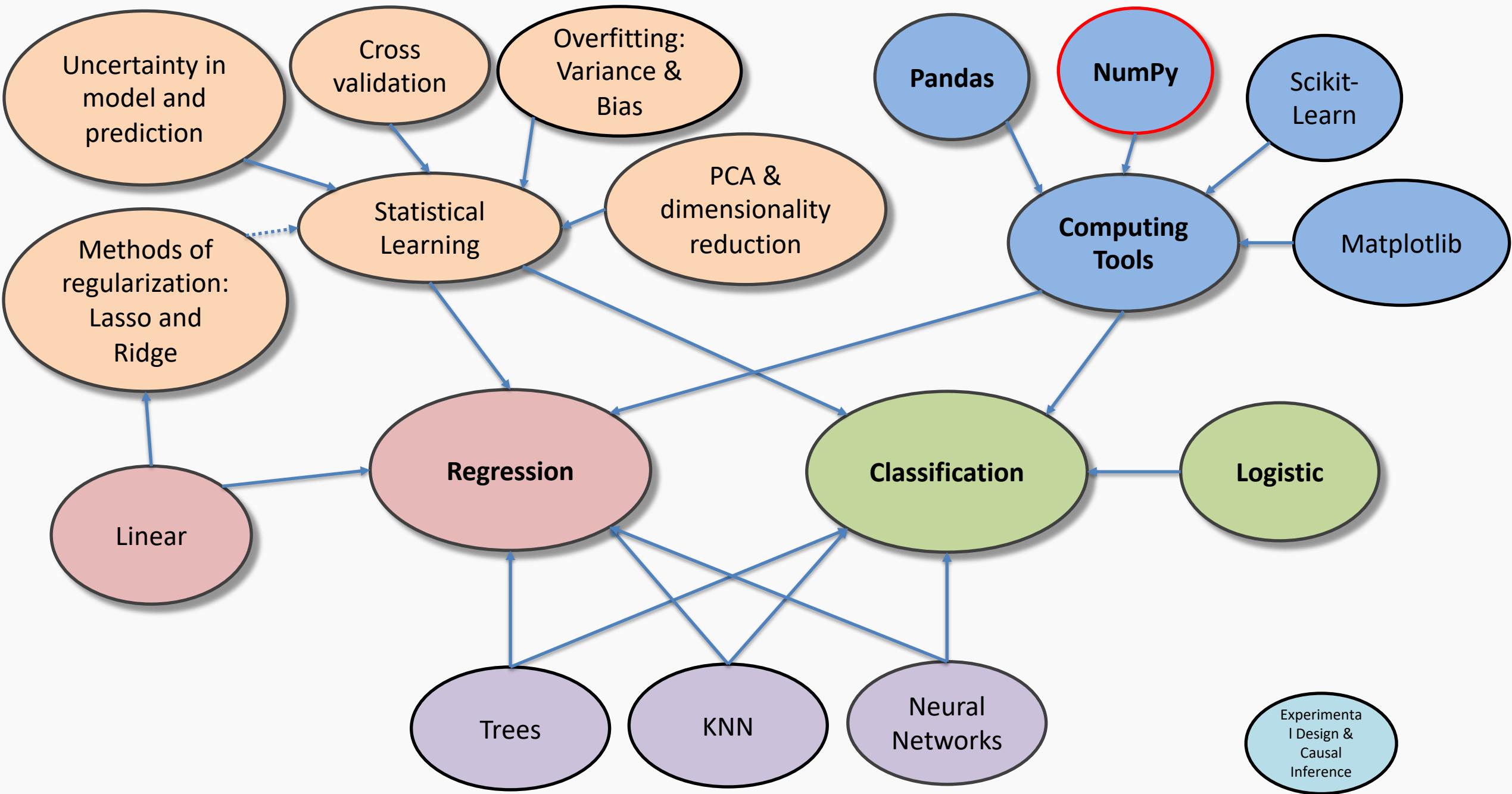
Arithmetic Operations with Fill Methods

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

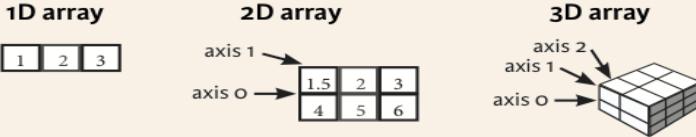
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

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



NumPy Arrays



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

<pre>>>> np.zeros((3,4))</pre>	Create an array of zeros
<pre>>>> np.ones((2,3),dtype=np.int16)</pre>	Create an array of ones
<pre>>>> d = np.arange(10,25,5)</pre>	Create an array of evenly spaced values (step value)
<pre>>>> np.linspace(0,2,9)</pre>	Create an array of evenly spaced values (number of samples)
<pre>>>> e = np.full((2,2),7)</pre>	Create a constant array
<pre>>>> f = np.eye(2)</pre>	Create a 2x2 identity matrix
<pre>>>> np.random.random((2,2))</pre>	Create an array with random values
<pre>>>> np.empty((3,2))</pre>	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

<pre>>>> np.int64</pre>	Signed 64-bit integer types
<pre>>>> np.float32</pre>	Standard double-precision floating point
<pre>>>> np.complex</pre>	Complex numbers represented by 128 floats
<pre>>>> np.bool</pre>	Boolean type storing TRUE and FALSE values
<pre>>>> np.object</pre>	Python object type
<pre>>>> np.string_</pre>	Fixed-length string type
<pre>>>> np.unicode_</pre>	Fixed-length unicode type

Inspecting Your Array

<pre>>>> a.shape</pre>	Array dimensions
<pre>>>> len(a)</pre>	Length of array
<pre>>>> b.ndim</pre>	Number of array dimensions
<pre>>>> e.size</pre>	Number of array elements
<pre>>>> b.dtype</pre>	Data type of array elements
<pre>>>> b.dtype.name</pre>	Name of data type
<pre>>>> b.astype(int)</pre>	Convert an array to a different type

Asking For Help

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

Array Mathematics

Arithmetic Operations

<pre>>>> g = a - b array([[-0.5, 0., 0.], [-3., -3., -3.]])</pre>	Subtraction
<pre>>>> np.subtract(a,b)</pre>	Subtraction
<pre>>>> b + a array([[2.5, 4., 6.], [5., 7., 9.]])</pre>	Addition
<pre>>>> np.add(b,a)</pre>	Addition
<pre>>>> a / b array([[0.66666667, 1., [0.25, 0.4, 0.5]]]]</pre>	Division
<pre>>>> np.divide(a,b)</pre>	Division
<pre>>>> a * b array([[1.5, 4., 9.], [4., 10., 18.]])</pre>	Multiplication
<pre>>>> np.multiply(a,b)</pre>	Multiplication
<pre>>>> np.exp(b)</pre>	Exponentiation
<pre>>>> np.sqrt(b)</pre>	Square root
<pre>>>> np.sin(a)</pre>	Print sines of an array
<pre>>>> np.cos(b)</pre>	Element-wise cosine
<pre>>>> np.log(a)</pre>	Element-wise natural logarithm
<pre>>>> e.dot(f) array([[7., 7.], [7., 7.]])</pre>	Dot product

Comparison

<pre>>>> a == b array([[False, True, True], [False, False, False]], dtype=bool)</pre>	Element-wise comparison
<pre>>>> a < 2 array([True, False, False], dtype=bool)</pre>	Element-wise comparison
<pre>>>> np.array_equal(a, b)</pre>	Array-wise comparison

Aggregate Functions

<pre>>>> a.sum()</pre>	Array-wise sum
<pre>>>> a.min()</pre>	Array-wise minimum value
<pre>>>> b.max(axis=0)</pre>	Maximum value of an array row
<pre>>>> b.cumsum(axis=1)</pre>	Cumulative sum of the elements
<pre>>>> a.mean()</pre>	Mean
<pre>>>> b.median()</pre>	Median
<pre>>>> a.corrcoef()</pre>	Correlation coefficient
<pre>>>> np.std(b)</pre>	Standard deviation

Copying Arrays

<pre>>>> h = a.view() >>> np.copy(a) >>> h = a.copy()</pre>	Create a view of the array with the same data
	Create a copy of the array
	Create a deep copy of the array

Sorting Arrays

<pre>>>> a.sort() >>> c.sort(axis=0)</pre>	Sort an array
	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting

<pre>>>> a[2] 3</pre>	1 2 3
<pre>>>> b[1,2] 6.0</pre>	1.5 2 3 4 5 6

Slicing

<pre>>>> a[0:2] array([1, 2])</pre>	1 2 3
<pre>>>> b[0:2,1] array([2., 5.])</pre>	1.5 2 3 4 5 6
<pre>>>> b[:,1] array([[1.5, 2., 3.]])</pre>	1.5 2 3 4 5 6
<pre>>>> c[1,:,:] array([[3., 2., 1.], [4., 5., 6.]])</pre>	1 2 3 4 5 6
<pre>>>> a[: :-1] array([3, 2, 1])</pre>	Reversed array a

Boolean Indexing

```
>>> a[a<2]
array([1])
```

<pre>>>> b[[1, 0, 1, 0], [0, 1, 2, 0]] array([4., 2., 6., 1.5])</pre>	1 2 3
<pre>>>> b[[1, 0, 1, 0], :, [0, 1, 2, 0]] array([[4., 5., 6., 4.], [1.5, 2., 3., 1.5], [4., 5., 6., 4.], [1.5, 2., 3., 1.5]])</pre>	Select elements (1,0),(0,1),(1,2) and (0,0)

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Permute array dimensions

Flatten the array

Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a, [1])
```

Return a new array with shape (2,6)

Append items to an array

Insert items in an array

Delete items from an array

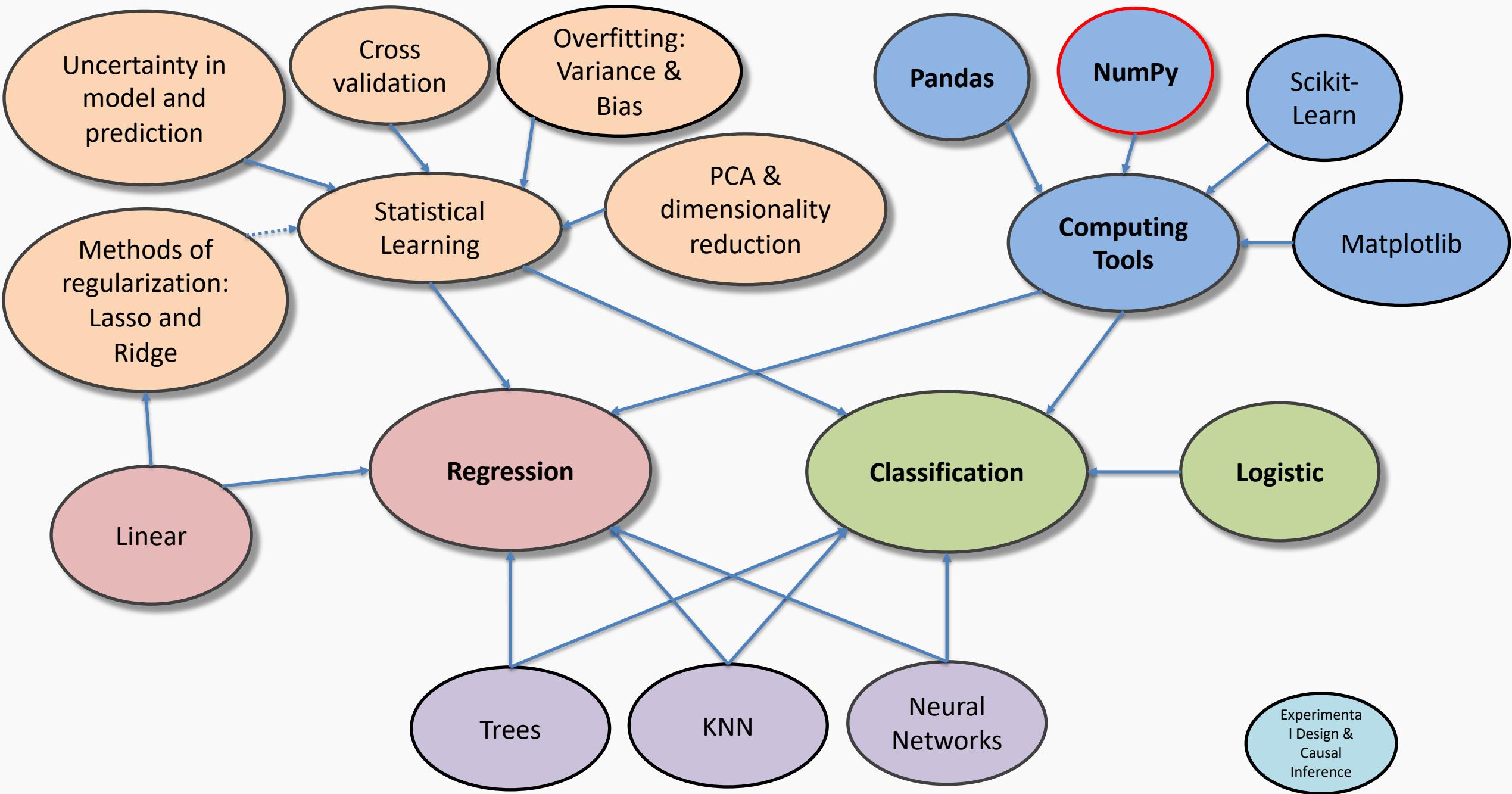
Combining Arrays

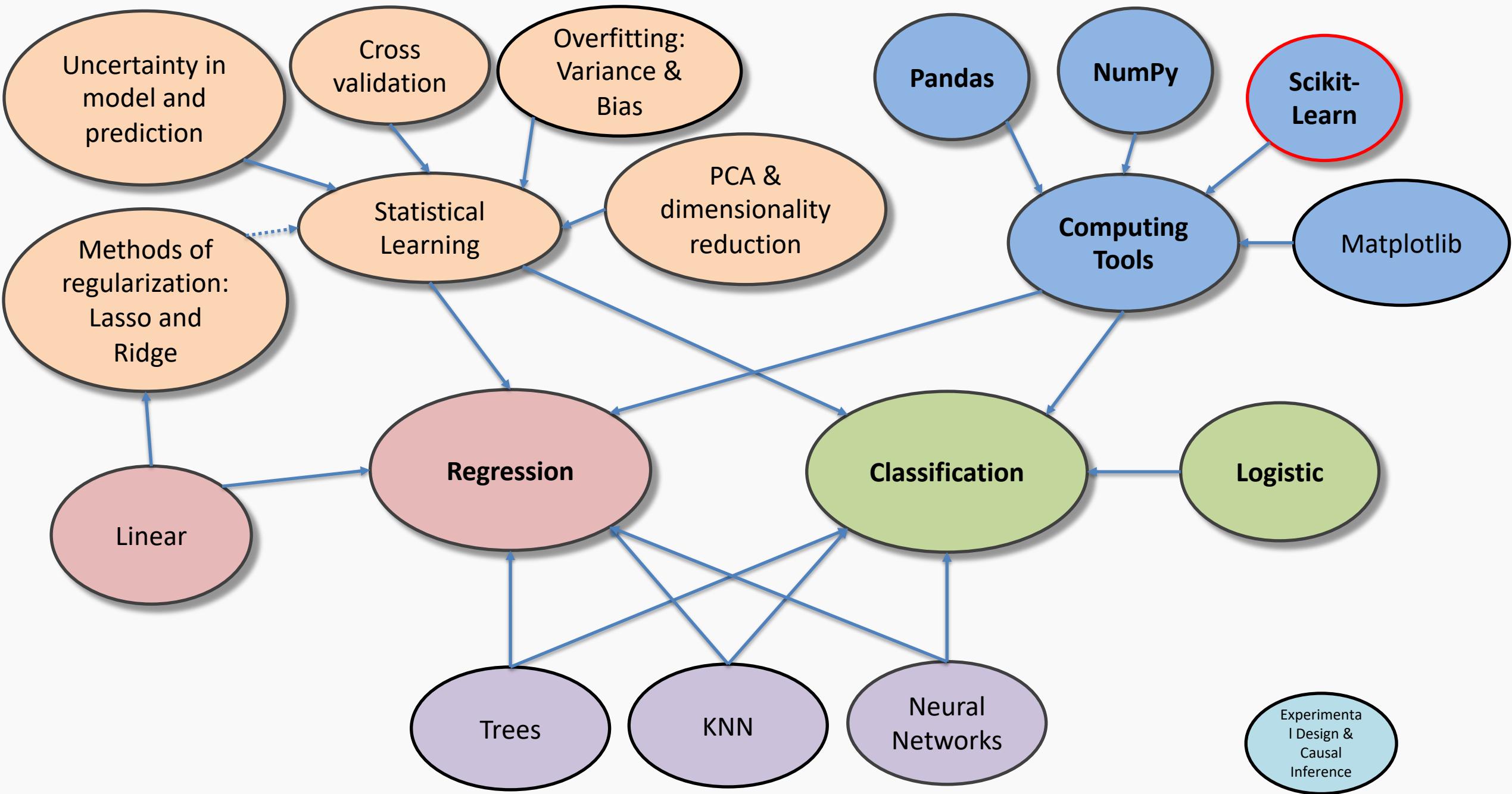
<pre>>>> np.concatenate((a,d),axis=0) array([1, 2, 3, 10, 15, 20])</pre>	Concatenate arrays
<pre>>>> np.vstack((a,b)) array([[1., 2., 3.], [1.5, 2., 3.], [4., 5., 6.]])</pre>	Stack arrays vertically (row-wise)
<pre>>>> np.r_[e,f] >>> np.hstack((e,f)) array([[7., 7., 1., 0.], [7., 7., 0., 1.]])</pre>	Stack arrays vertically (row-wise)
<pre>>>> np.hstack((e,f)) array([[7., 7., 1., 0.], [7., 7., 0., 1.]])</pre>	Stack arrays horizontally (column-wise)
<pre>>>> np.column_stack((a,d)) array([[1, 10], [2, 15], [3, 20]])</pre>	Create stacked column-wise arrays
<pre>>>> np.c_[a,d]</pre>	Create stacked column-wise arrays

Splitting Arrays

<pre>>>> np.hsplit(a,3) [array([1]),array([2]),array([3])]</pre>	Split the array horizontally at the 3rd index
<pre>>>> np.vsplit(c,2) [array([[1.5, 2., 1.], [4., 5., 6.]]), array([[3., 2., 1.], [4., 5., 6.]]])</pre>	Split the array vertically at the 2nd index







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]	Create a dense meshgrid
>>> np.ogrid[0:2,0:2]	Create an open meshgrid
>>> np.r_[3,[0]*5,-1:1:10j]	Stack arrays vertically (row-wise)
>>> np.c_[b,c]	Create stacked column-wise arrays

Shape Manipulation

>>> np.transpose(b)	Permute array dimensions
>>> b.flatten()	Flatten the array
>>> np.hstack((b,c))	Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))	Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)	Split the array horizontally at the 2nd index
>>> np.vsplit(d,2)	Split the array vertically at the 2nd index

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])
```

Create a polynomial object

Vectorizing Functions

```
>>> def myfunc(a):
...     if a < 0:
...         return a**2
...     else:
...         return a/2
>>> np.vectorize(myfunc)
```

Vectorize functions

Type Handling

>>> np.real(b)	Return the real part of the array elements
>>> np.imag(b)	Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)	Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)	Cast object to a data type

Other Useful Functions

>>> np.angle(b,deg=True)	Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)	Create an array of evenly spaced values (number of samples)
>>> g[3:] += np.pi	Unwrap
>>> np.unwrap(g)	Create an array of evenly spaced values (log scale)
>>> np.logspace(0,10,3)	Return values from a list of arrays depending on conditions
>>> np.select([c<4],[c*2])	Factorial
>>> misc.factorial(a)	Combine N things taken at k time
>>> misc.comb(10,3,exact=True)	Weights for N-point central derivative
>>> misc.central_diff_weights(3)	Find the n-th derivative of a function at a point
>>> misc.derivative(myfunc,1.0)	

Linear Algebra

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

Also see NumPy

```
>>> 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)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.isspmatrix_csc(A)
```

Sparse Matrix Routines

Inverse

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

Norm

```
>>> sparse.linalg.norm(I)
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)      Sparse matrix exponential
```

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Matrix Functions

Addition

```
>>> np.add(A,D)
```

Subtraction

```
>>> np.subtract(A,D)
```

Division

```
>>> np.divide(A,D)
```

Multiplication

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

```
>>> np.multiply(D,A)
```

```
>>> np.dot(A,D)
```

```
>>> np.vdot(A,D)
```

```
>>> np.inner(A,D)
```

```
>>> np.outer(A,D)
```

```
>>> np.tensordot(A,D)
```

```
>>> np.kron(A,D)
```

Exponential Functions

```
>>> linalg.expm(A)
```

```
>>> linalg.expm2(A)
```

```
>>> linalg.expm3(D)
```

Logarithm Function

```
>>> linalg.logm(A)
```

Trigonometric Functions

```
>>> linalg.sinm(D)
```

```
>>> linalg.cosm(D)
```

```
>>> linalg.tanm(A)
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
```

```
>>> linalg.coshm(D)
```

```
>>> linalg.tanhm(A)
```

Matrix Sign Function

```
>>> np.signm(A)
```

Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x**x)
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
```

```
>>> l1, l2 = la
```

```
>>> v[:,0]
```

```
>>> v[:,1]
```

```
>>> linalg.eigvals(A)
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
```

```
>>> M,N = B.shape
```

```
>>> Sig = linalg.diagsvd(s,M,N)
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)
```

```
>>> sparse.linalg.svds(H, 2)
```

Addition

Subtraction

Division

Multiplication operator

Multiplication

Dot product

Vector dot product

Inner product

Outer product

Tensor dot product

Kronecker product

Matrix exponential

Matrix exponential (Taylor Series)

Matrix exponential (eigenvalue decomposition)

Matrix logarithm

Matrix sine

Matrix cosine

Matrix tangent

Hyperbolic matrix sine

Hyperbolic matrix cosine

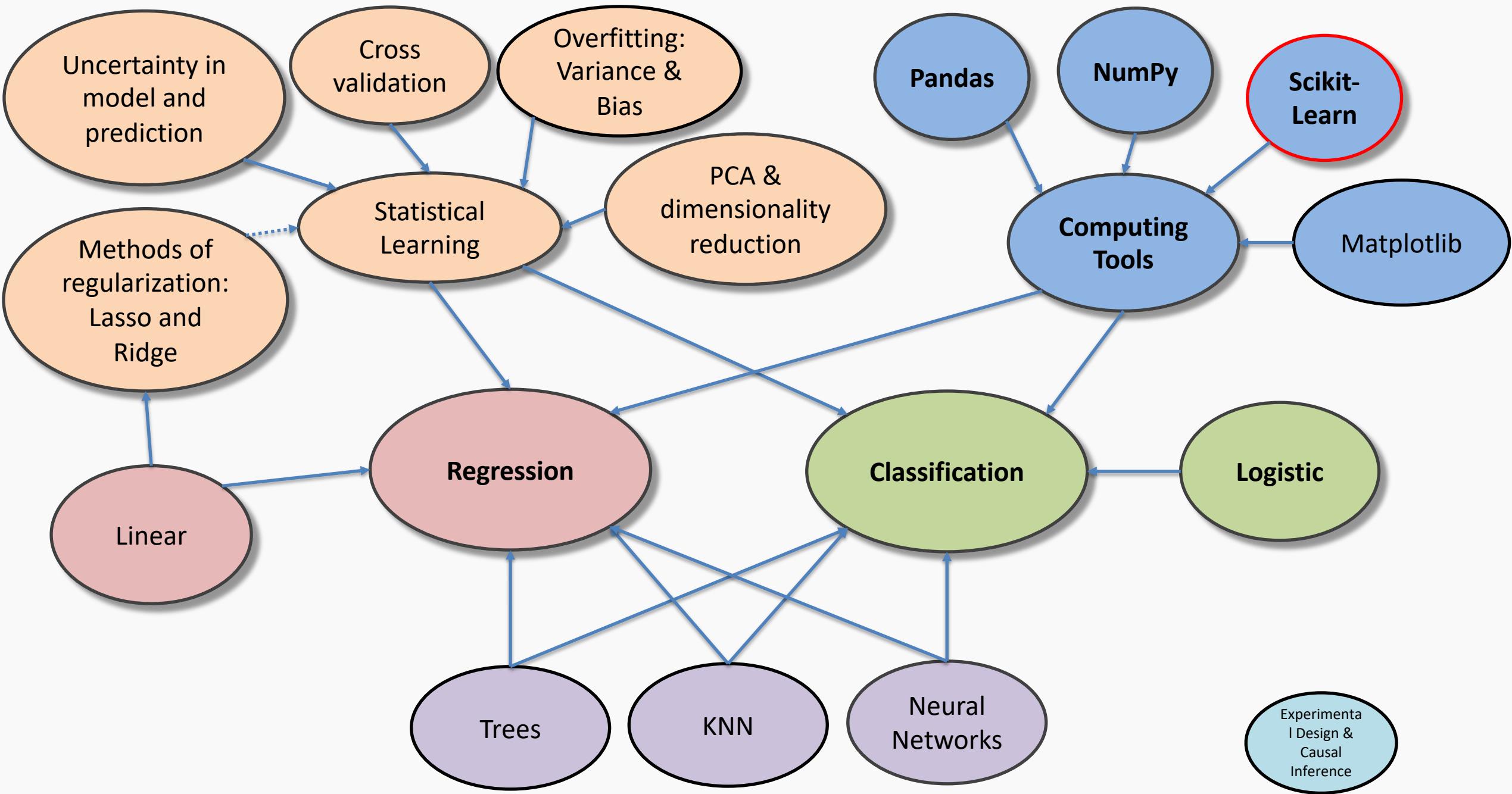
Hyperbolic matrix tangent

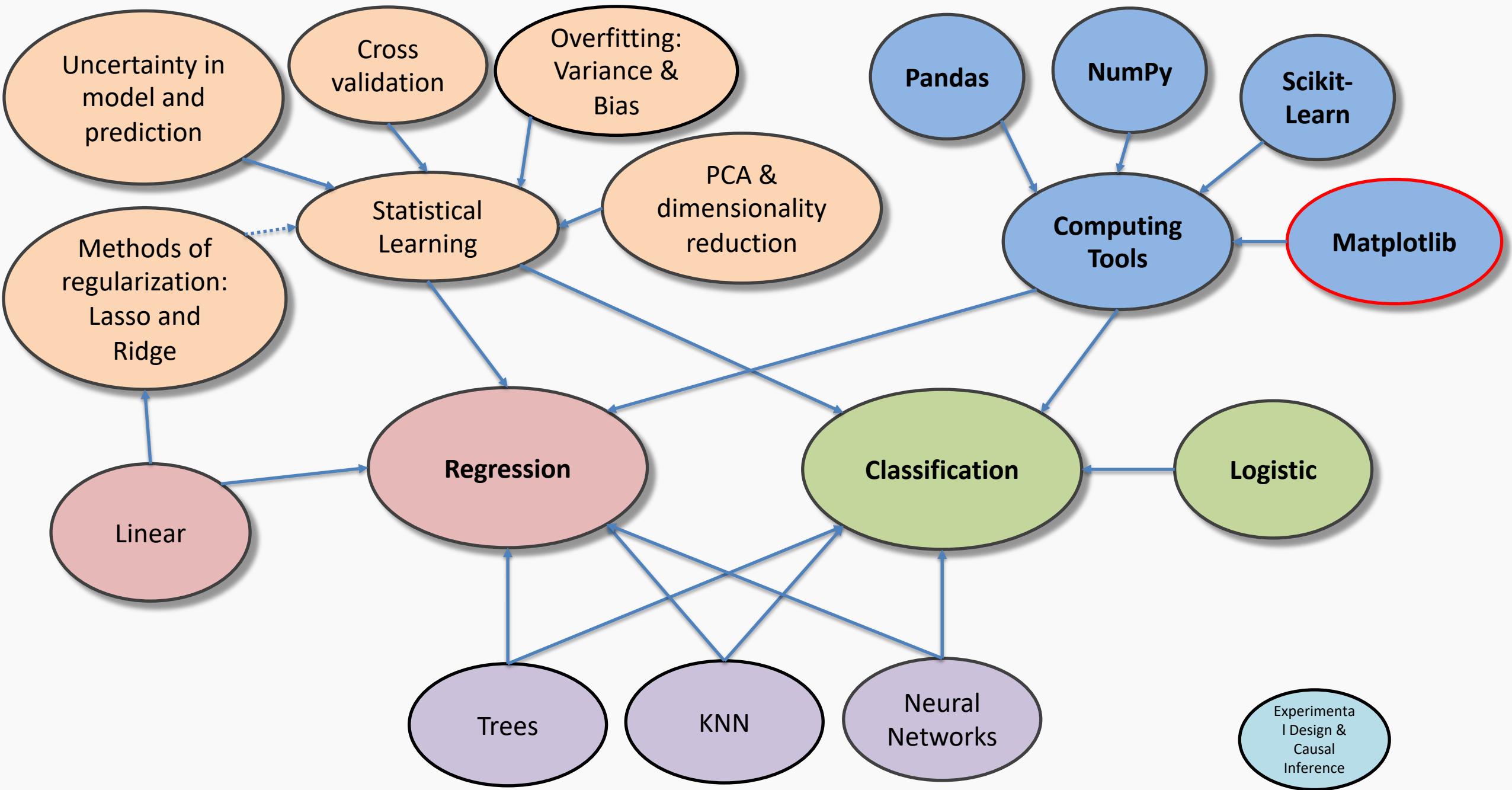
Matrix sign function

Matrix square root

Evaluate matrix function







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))
```

Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()  
>>> ax1 = fig.add_subplot(221) # row-col-num  
>>> ax3 = fig.add_subplot(212)  
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)  
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()  
>>> lines = ax.plot(x,y)  
>>> ax.scatter(x,y)  
>>> axes[0,0].bar([1,2,3],[3,4,5])  
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])  
>>> axes[1,1].axhline(0.45)  
>>> axes[0,1].axvline(0.65)  
>>> ax.fill(x,y,color='blue')  
>>> ax.fill_between(x,y,color='yellow')
```

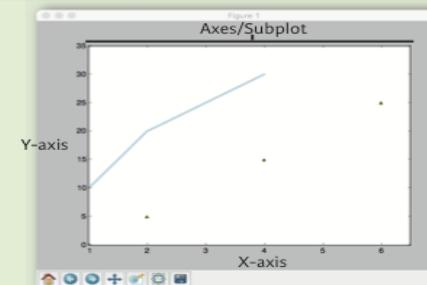
2D Data or Images

```
>>> fig, ax = plt.subplots()  
>>> im = ax.imshow(img,  
                  cmap='gist_earth',  
                  interpolation='nearest',  
                  vmin=-2,  
                  vmax=2)
```

Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
- 2 Create plot
- 3 Plot
- 4 Customize plot
- 5 Save plot
- 6 Show plot

```
>>> import matplotlib.pyplot as plt  
>>> x = [1,2,3,4] Step 1  
>>> y = [10,20,25,30] Step 2  
>>> fig = plt.figure() Step 3  
>>> ax = fig.add_subplot(111) Step 3  
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3, 4  
>>> ax.scatter([2,4,6],  
             [5,15,25],  
             color='darkgreen',  
             marker='^')  
>>> ax.set_xlim(1, 6.5)  
>>> plt.savefig('foo.png')  
>>> plt.show() Step 6
```

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

```
>>> ax.text(1,-2.1,  
           'Example Graph',  
           style='italic')  
>>> ax.annotate("Sine",  
               xy=(8, 0),  
               xycoords='data',  
               xytext=(10.5, 0),  
               textcoords='data',  
               arrowprops=dict(arrowstyle="->",  
                               connectionstyle="arc3"),  
               rotation=90)
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)  
>>> axes[1,1].quiver(y,z)  
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)  
>>> ax3.boxplot(y)  
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

Mathtext

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

```
>>> ax.margins(x=0.0,y=0.1)  
>>> ax.axis('equal')  
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])  
>>> ax.set_xlim(0,10.5)
```

Legends

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

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),  
                ticklabels=[3,100,-12,"foo"])  
>>> ax.tick_params(axis='y',  
                           direction='inout',  
                           length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,  
                        hspace=0.3,  
                        left=0.125,  
                        right=0.9,  
                        top=0.9,  
                        bottom=0.1)
```

```
>>> fig.tight_layout()
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)  
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible
Move the bottom axis line outward

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

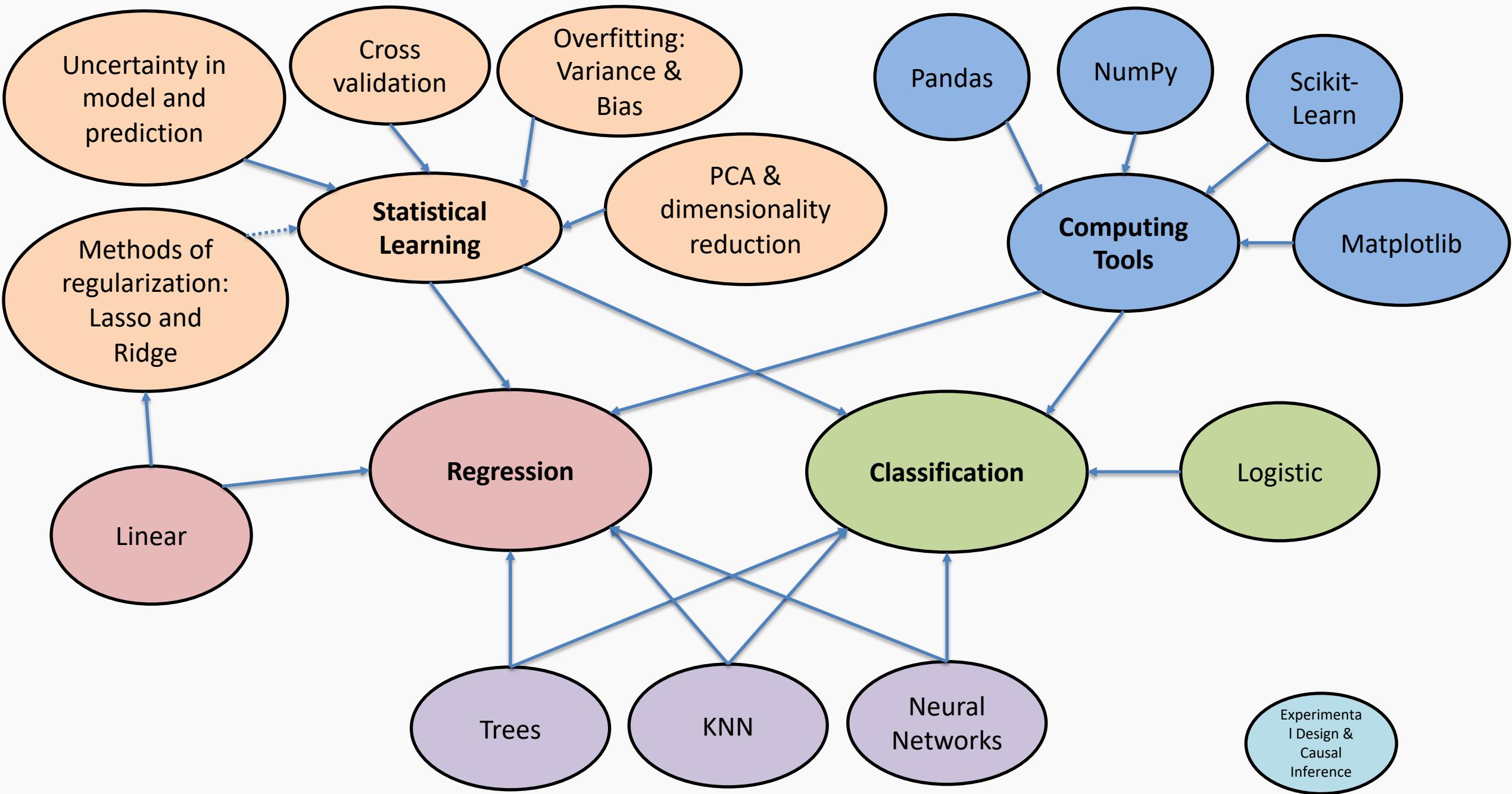
```
>>> plt.show()
```

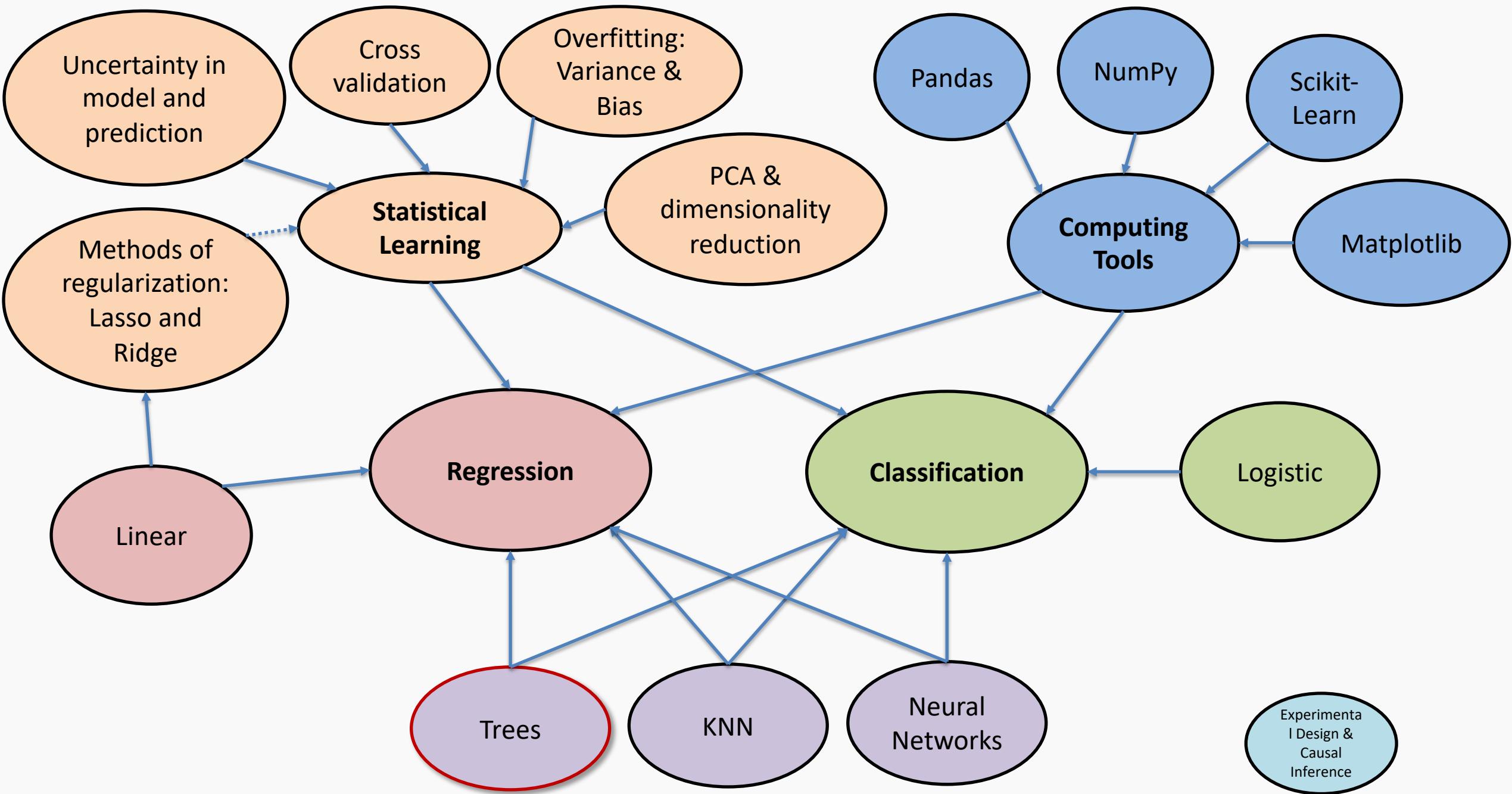
Close & Clear

```
>>> plt.clf()  
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window

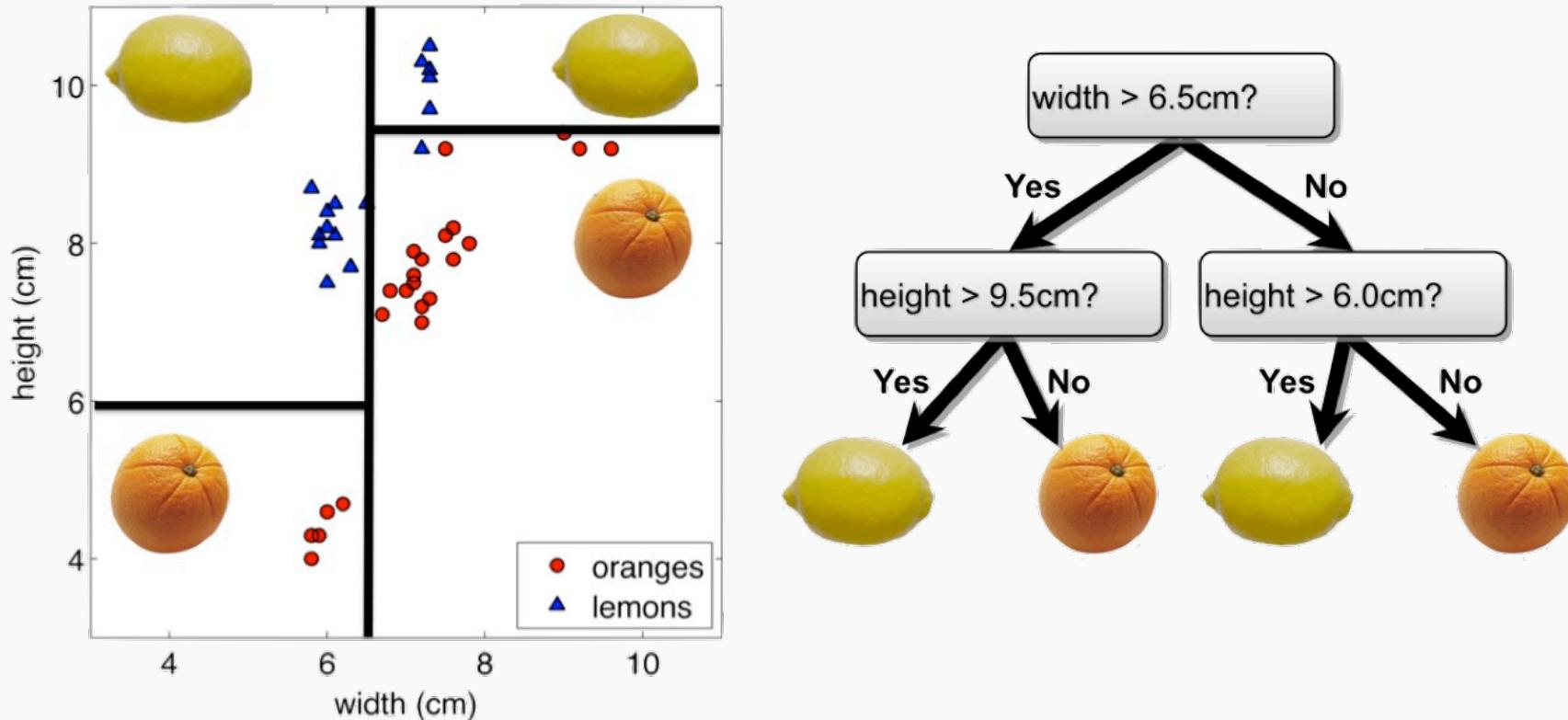




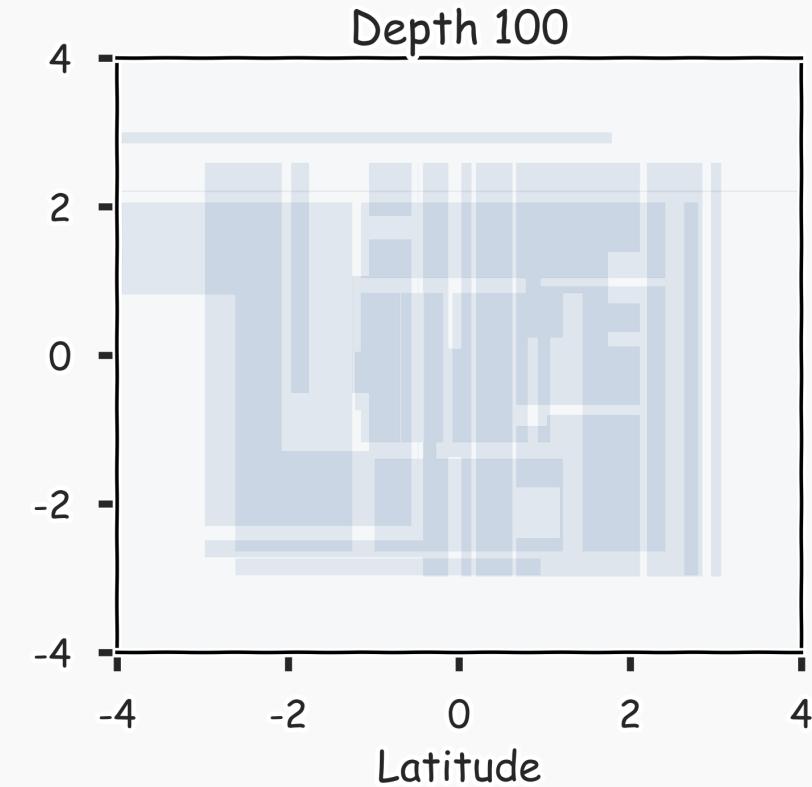
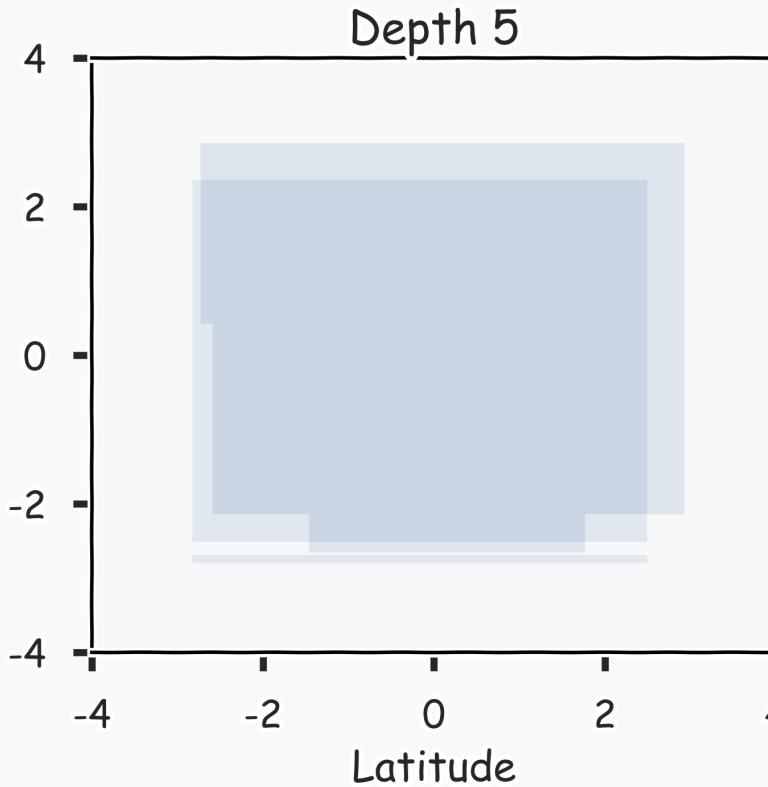
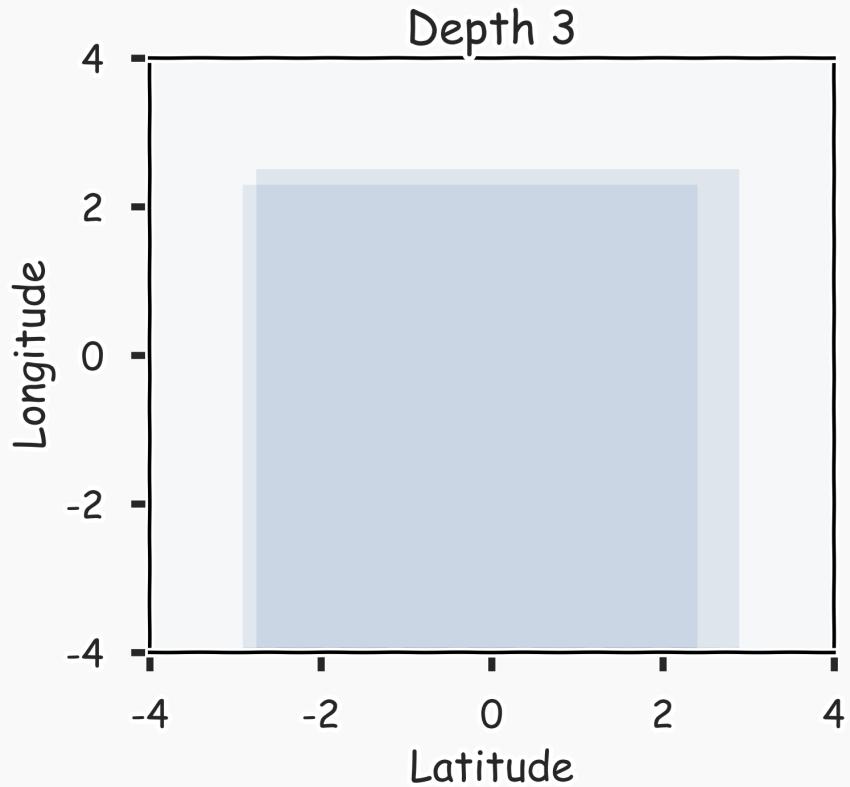


The Geometry of Flow Charts

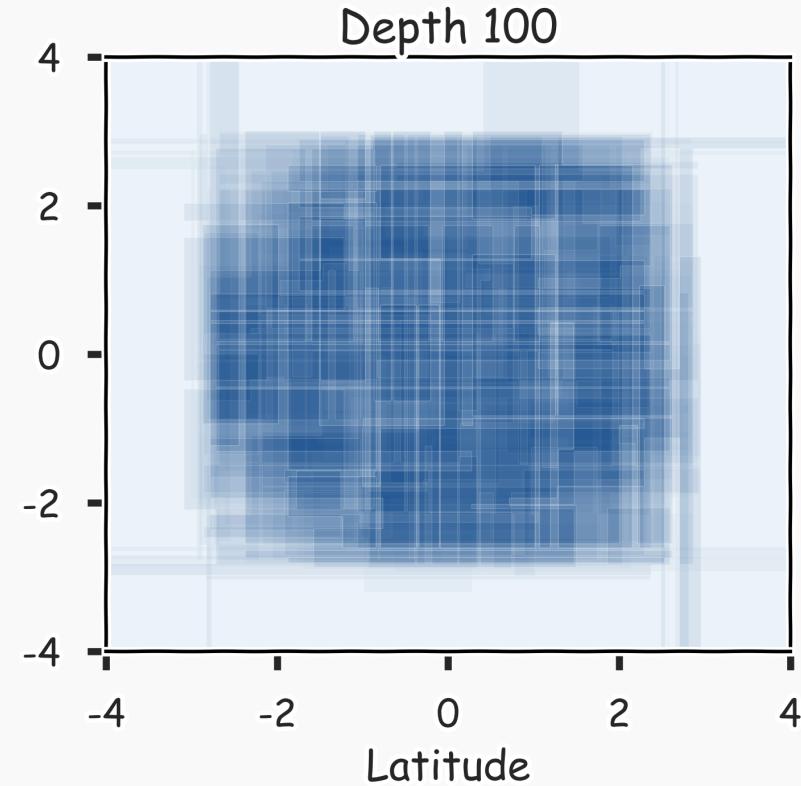
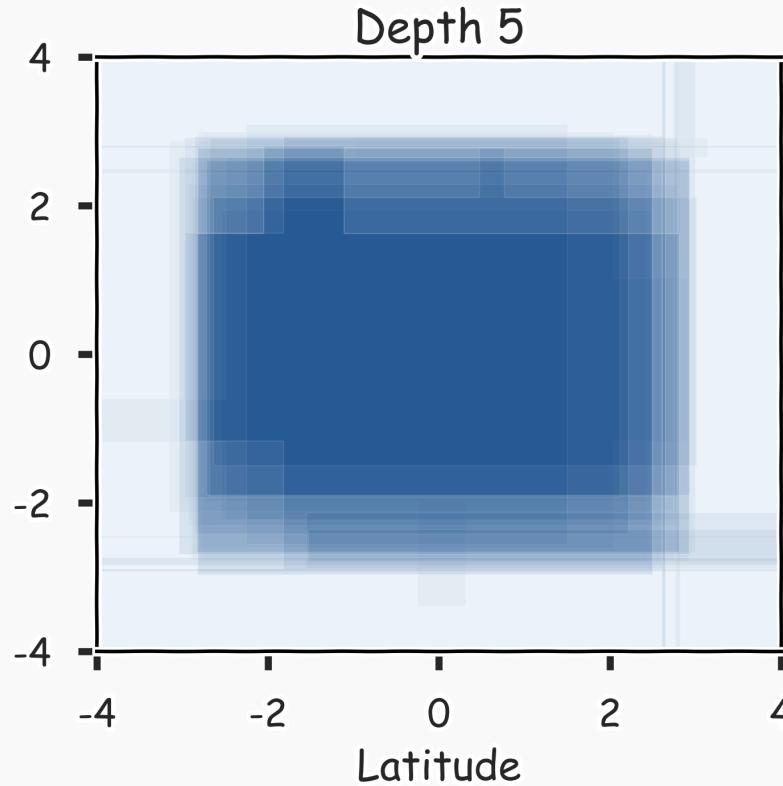
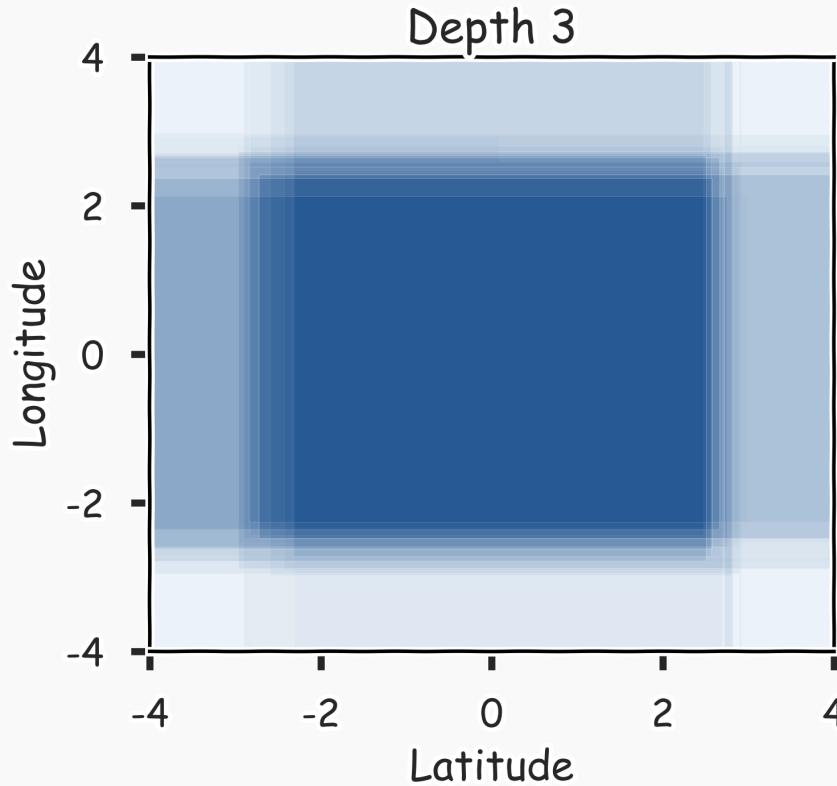
Each comparison and branching represents splitting a region in the feature space on a single feature. Typically, at each iteration, we split once along one dimension (one predictor). Why?



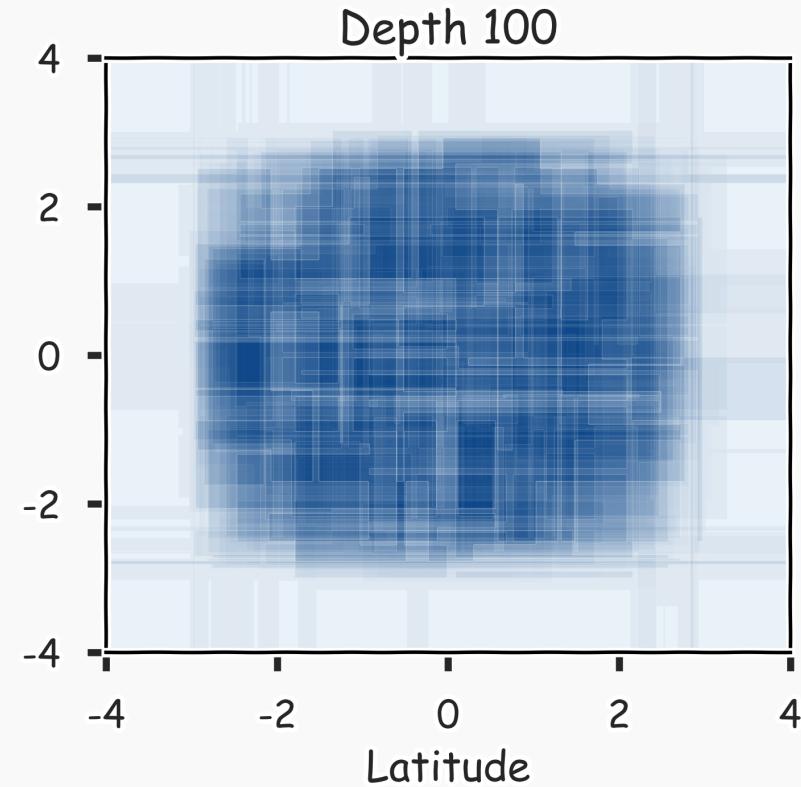
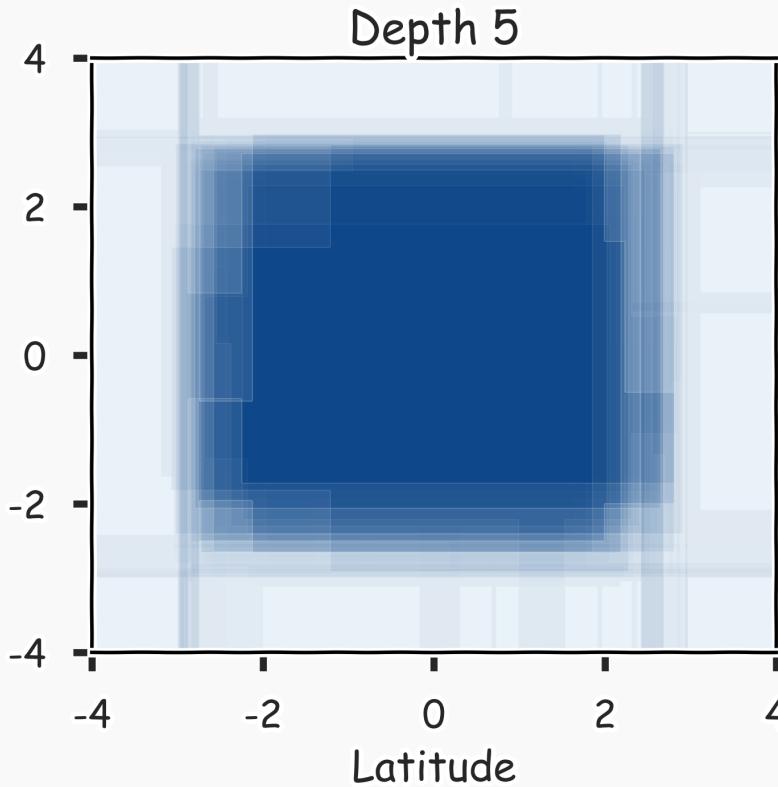
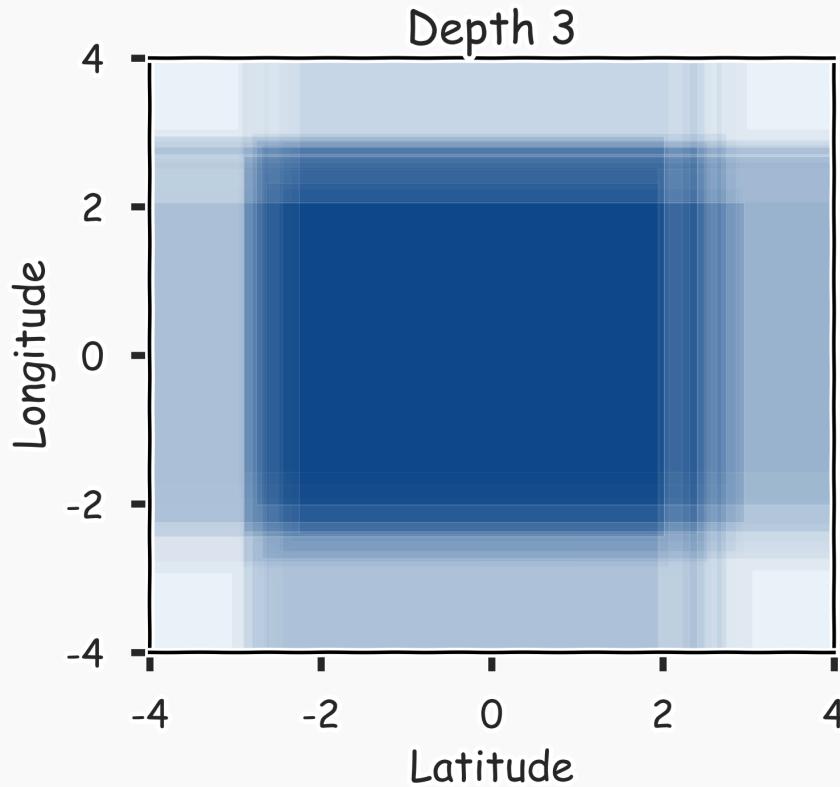
Combine them? 2 magic realisms



Combine them? 20 magic realisms



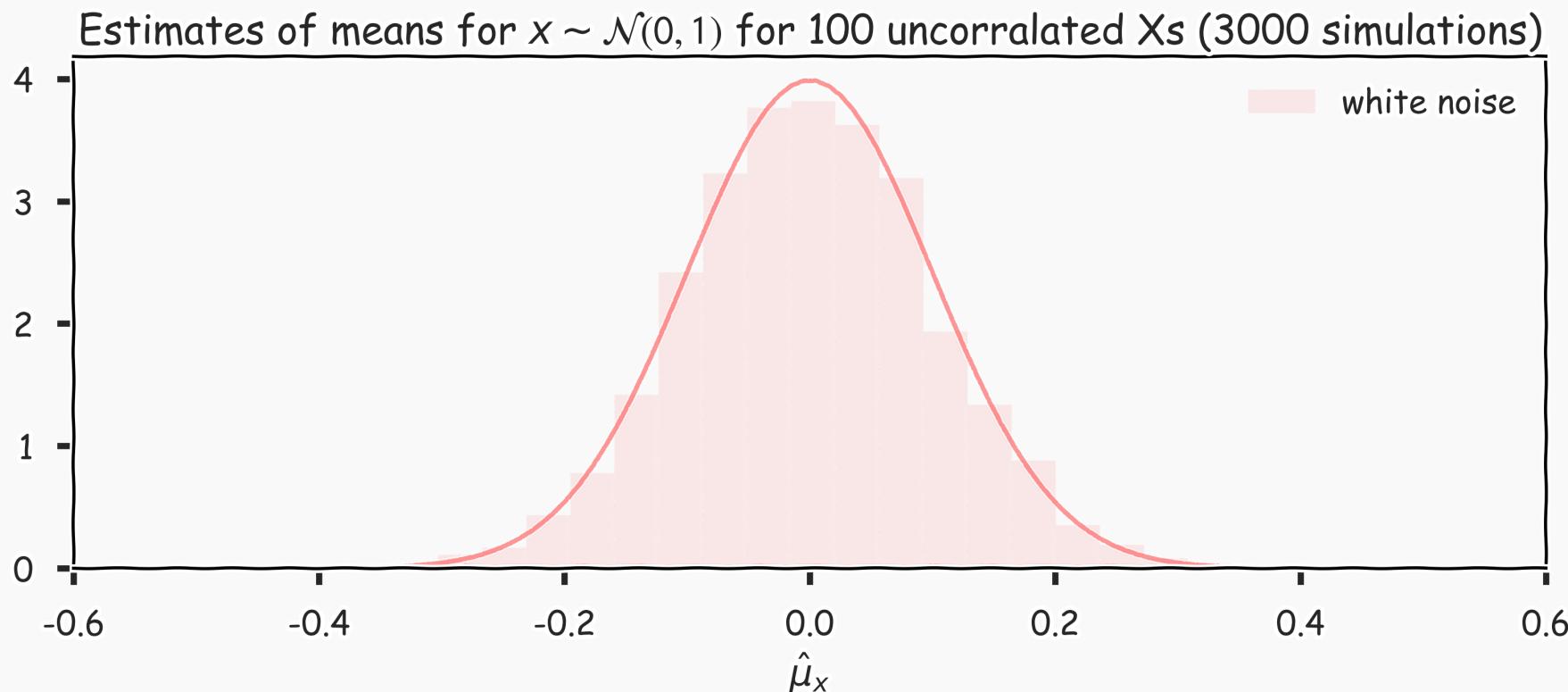
Combine them? 100 magic realisms



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 :

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

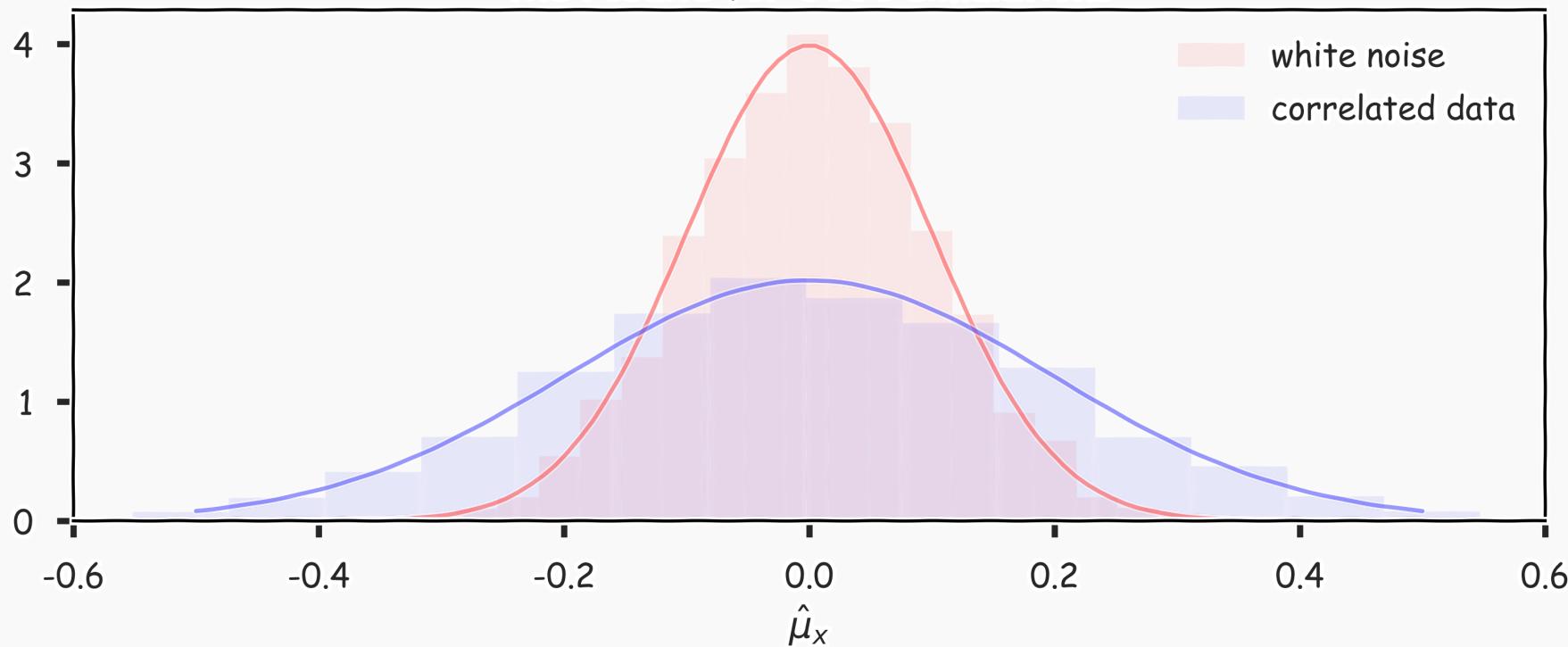


Improving on Bagging

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

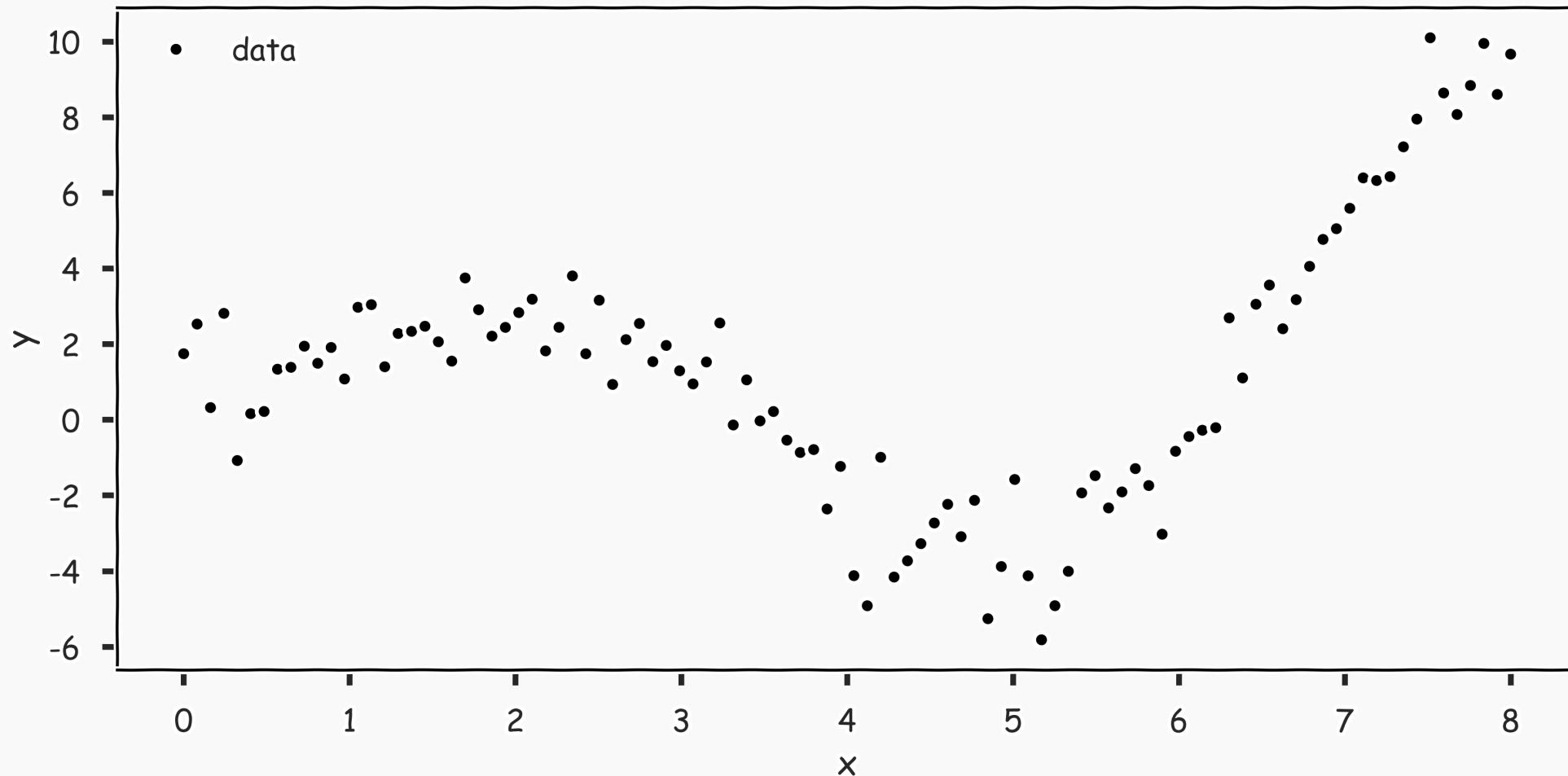
Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

To de-correlate the trees, we:

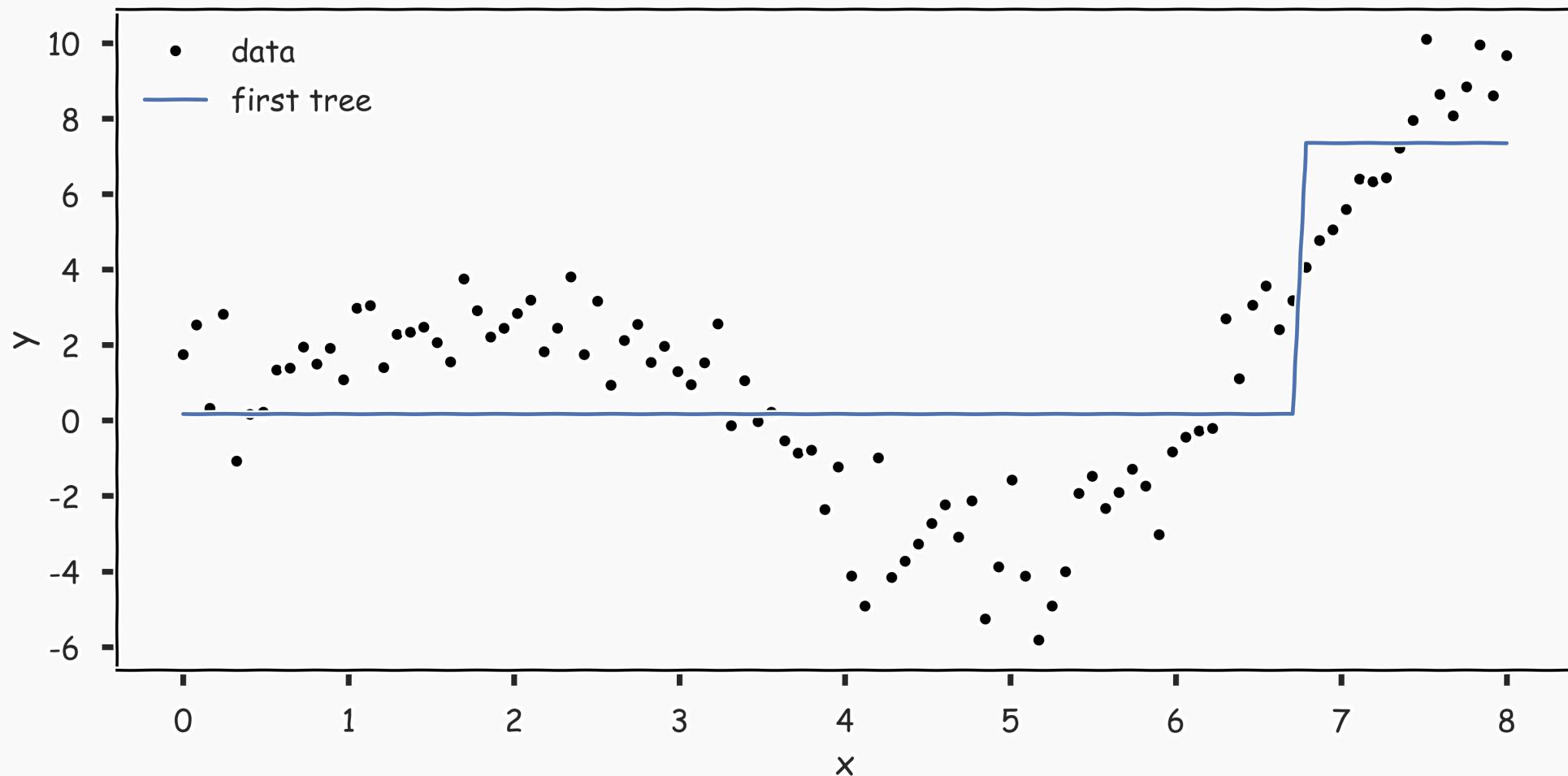
1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)
2. for each tree, at each split, we ***randomly*** select a set of J' predictors from the full set of predictors.

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

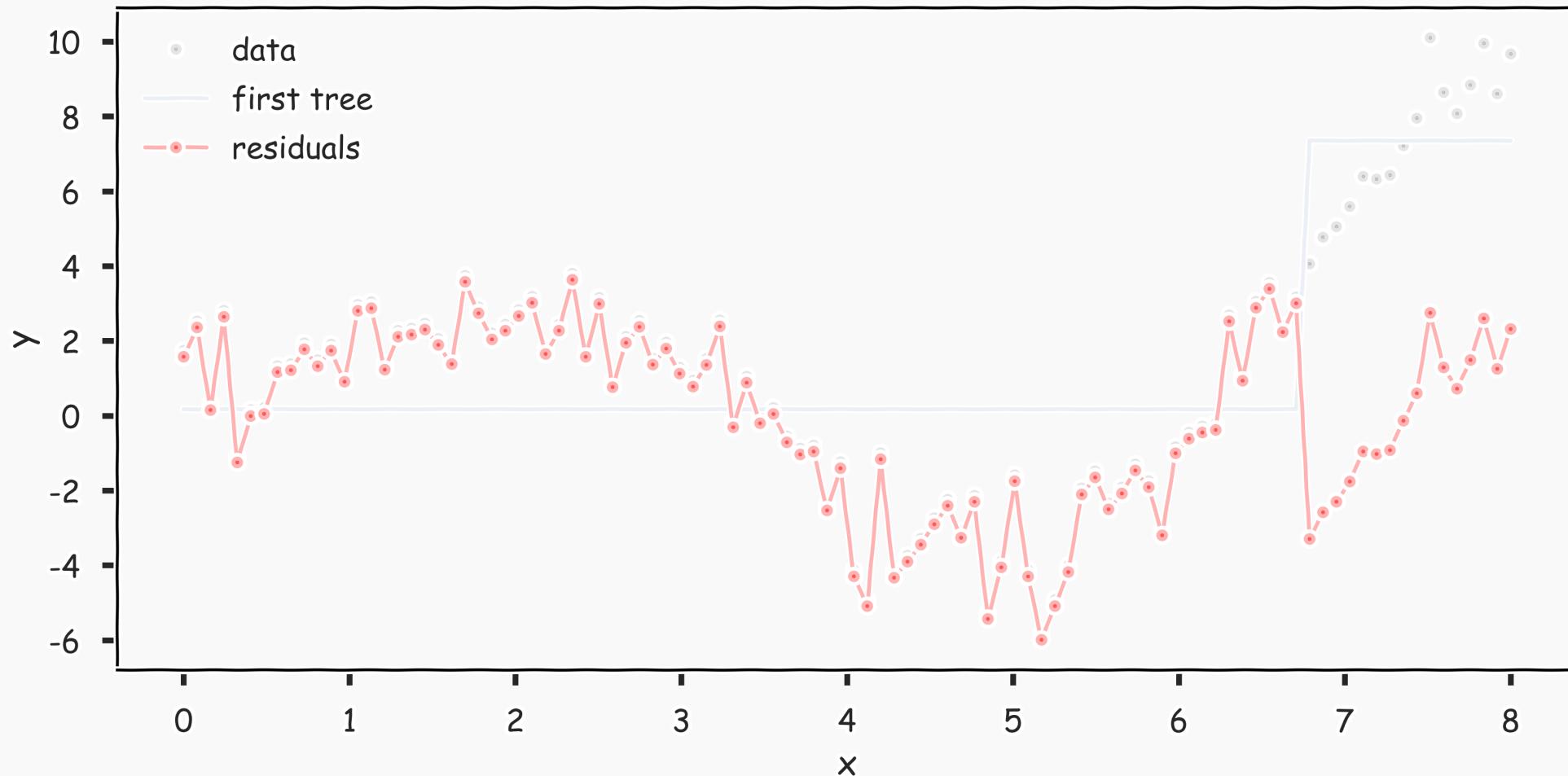
Gradient Boosting: illustration



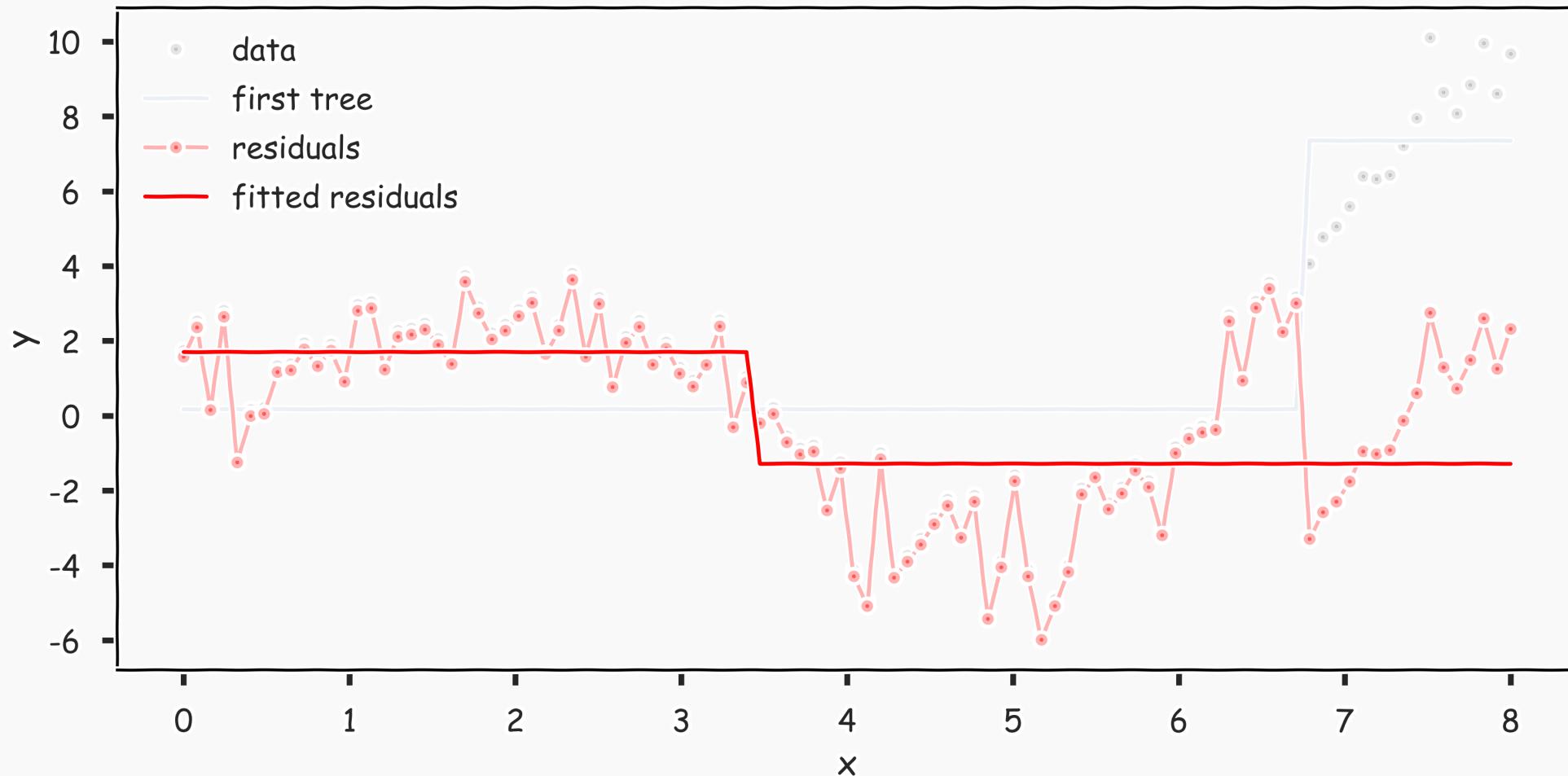
Gradient Boosting: illustration



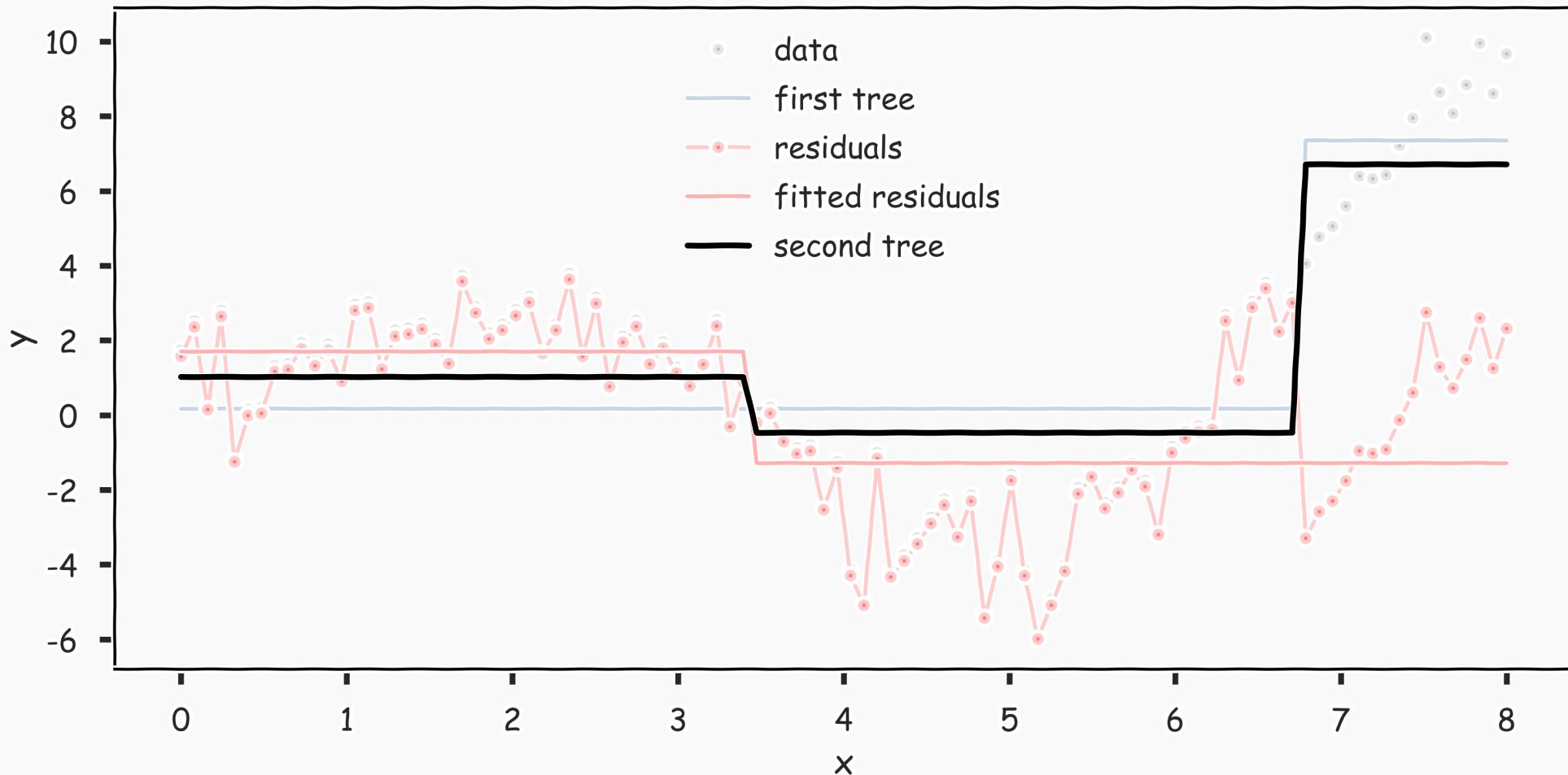
Gradient Boosting: illustration



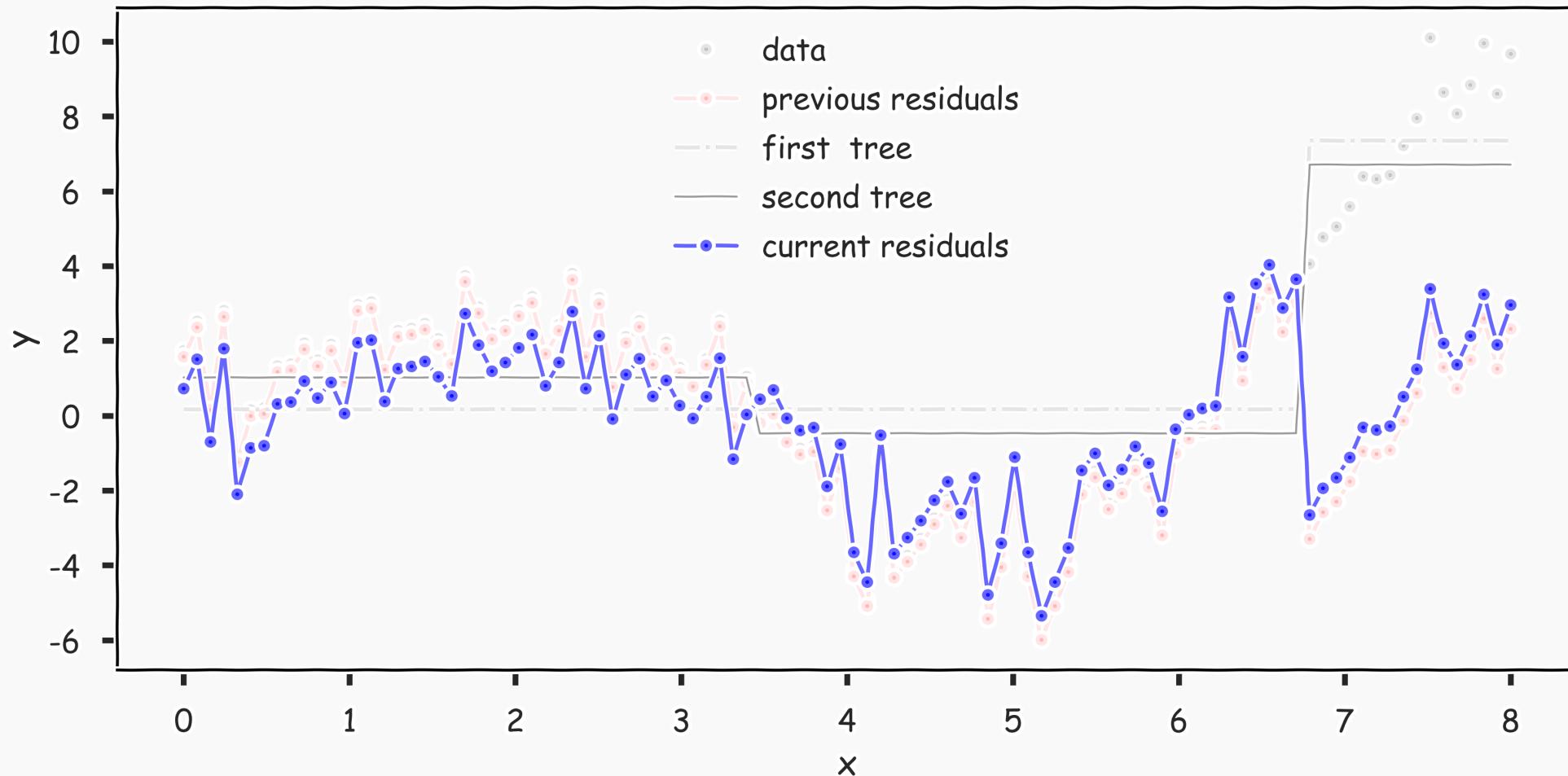
Gradient Boosting: illustration



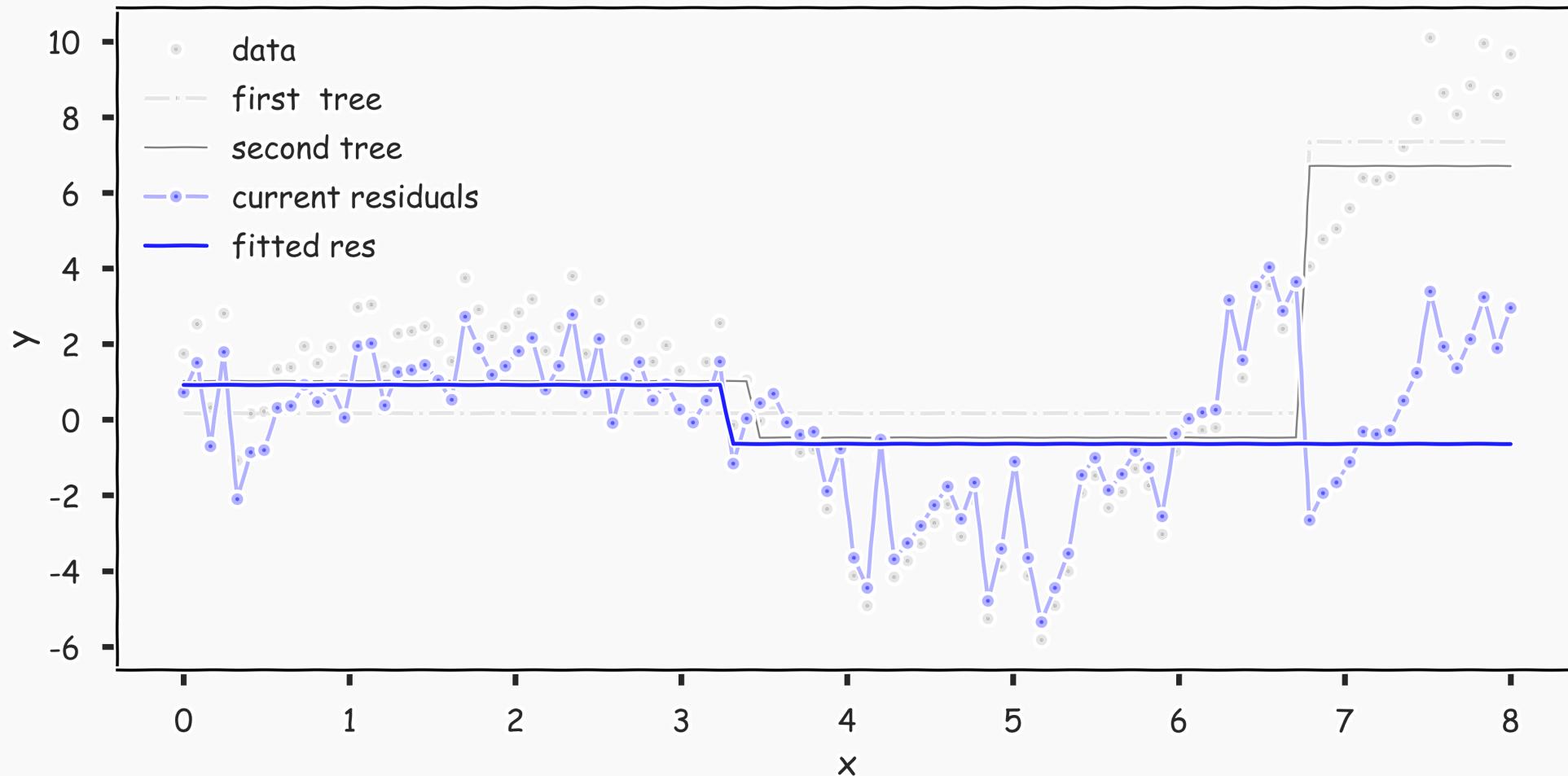
Gradient Boosting: illustration



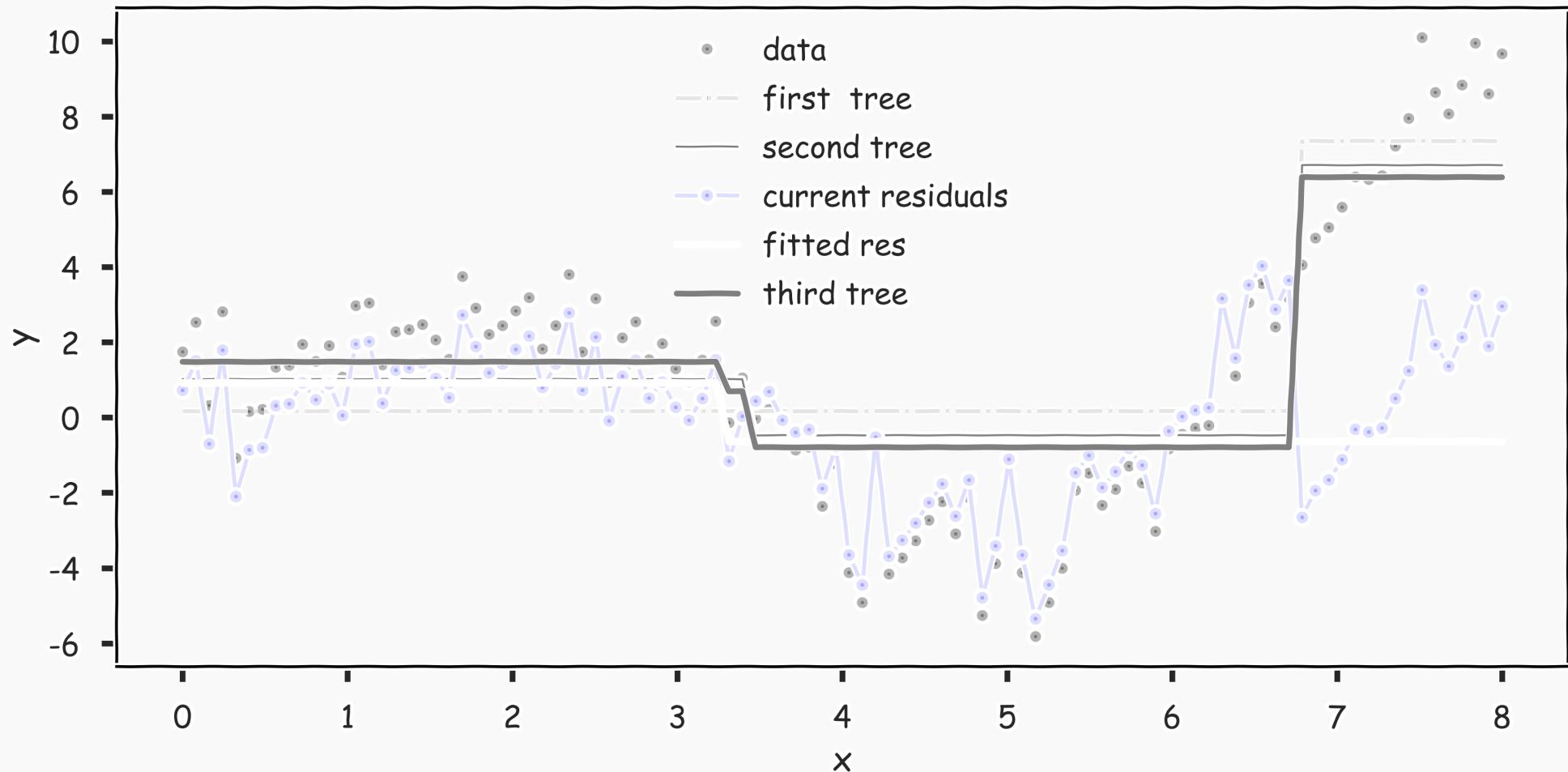
Gradient Boosting: illustration

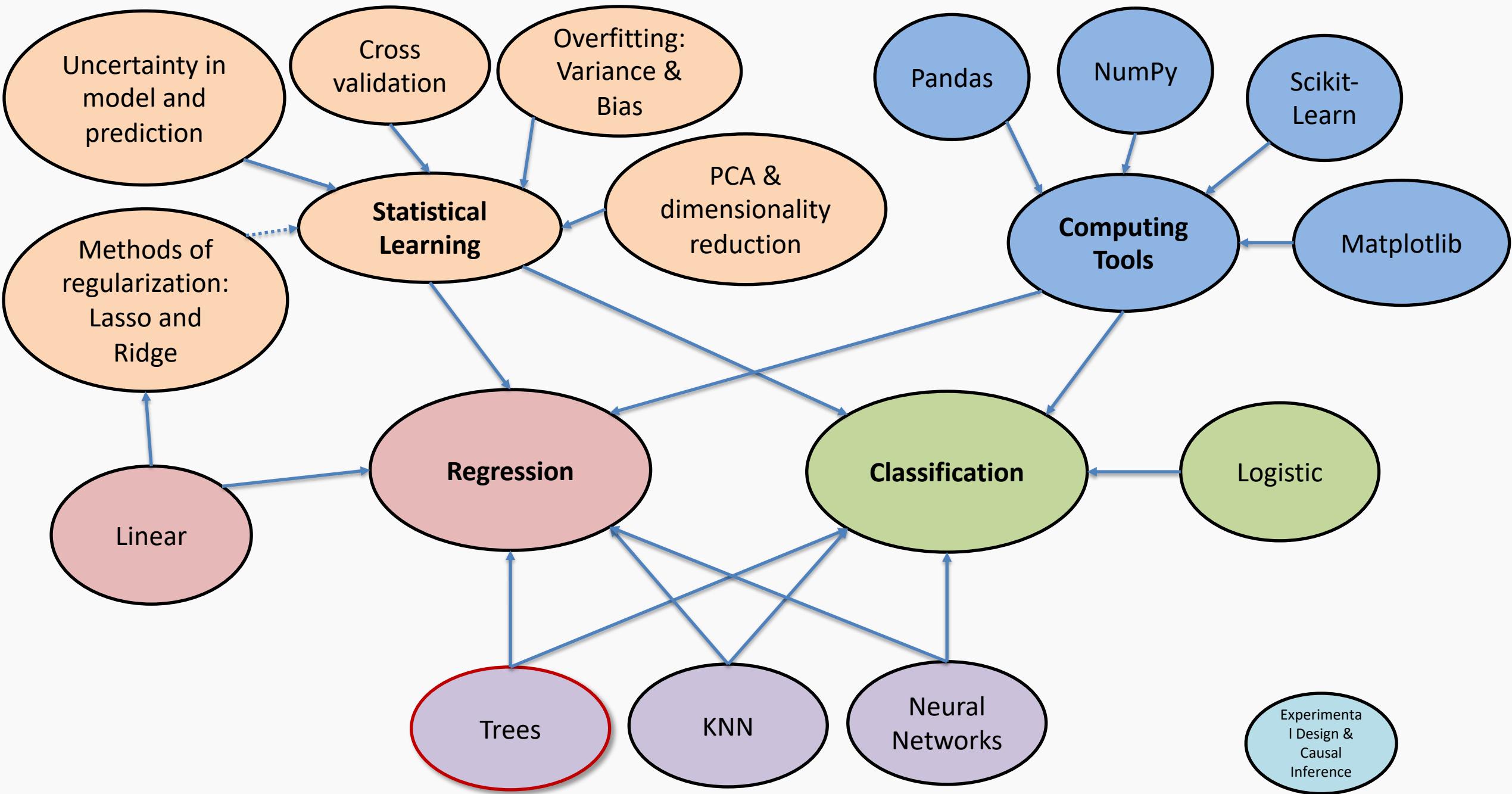


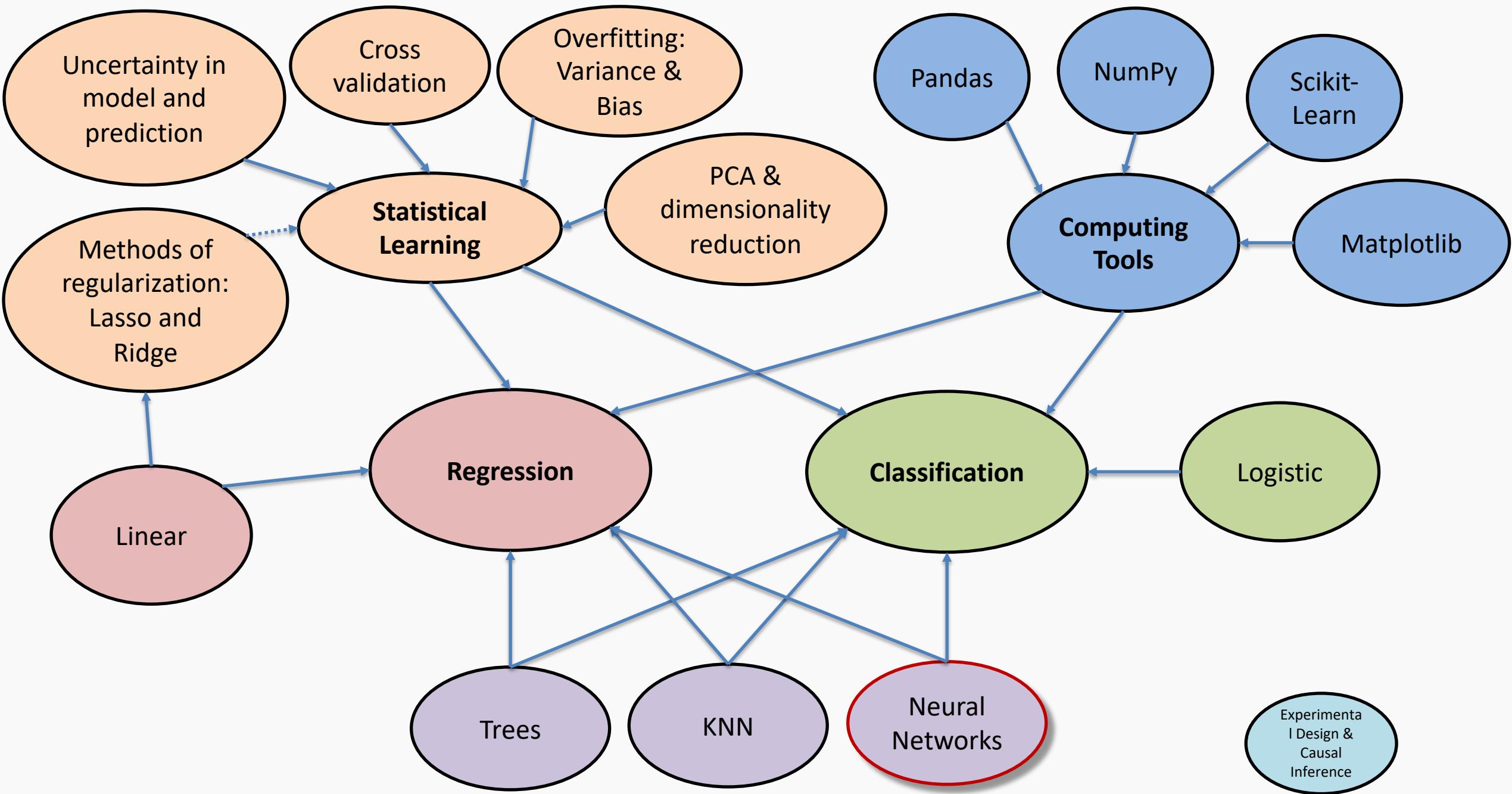
Gradient Boosting: illustration



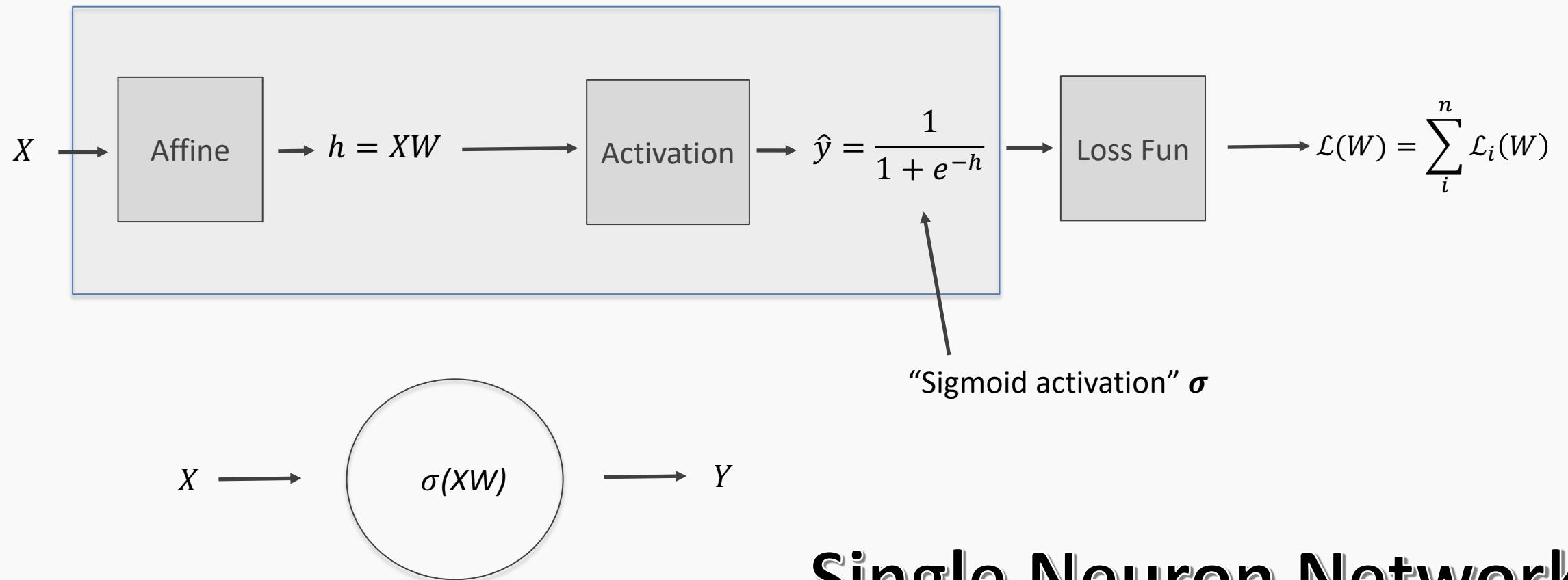
Gradient Boosting: illustration





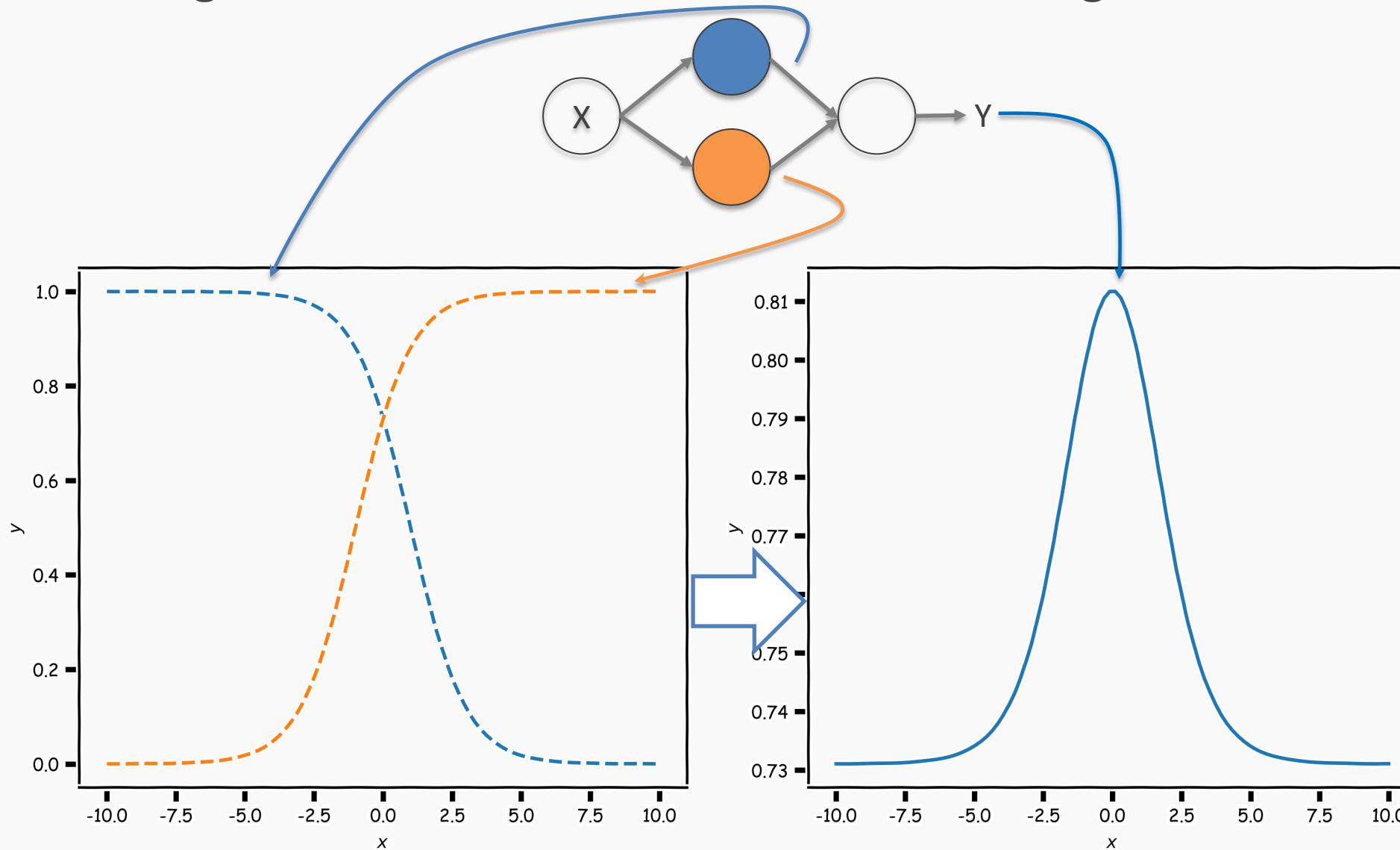


Build our first ANN

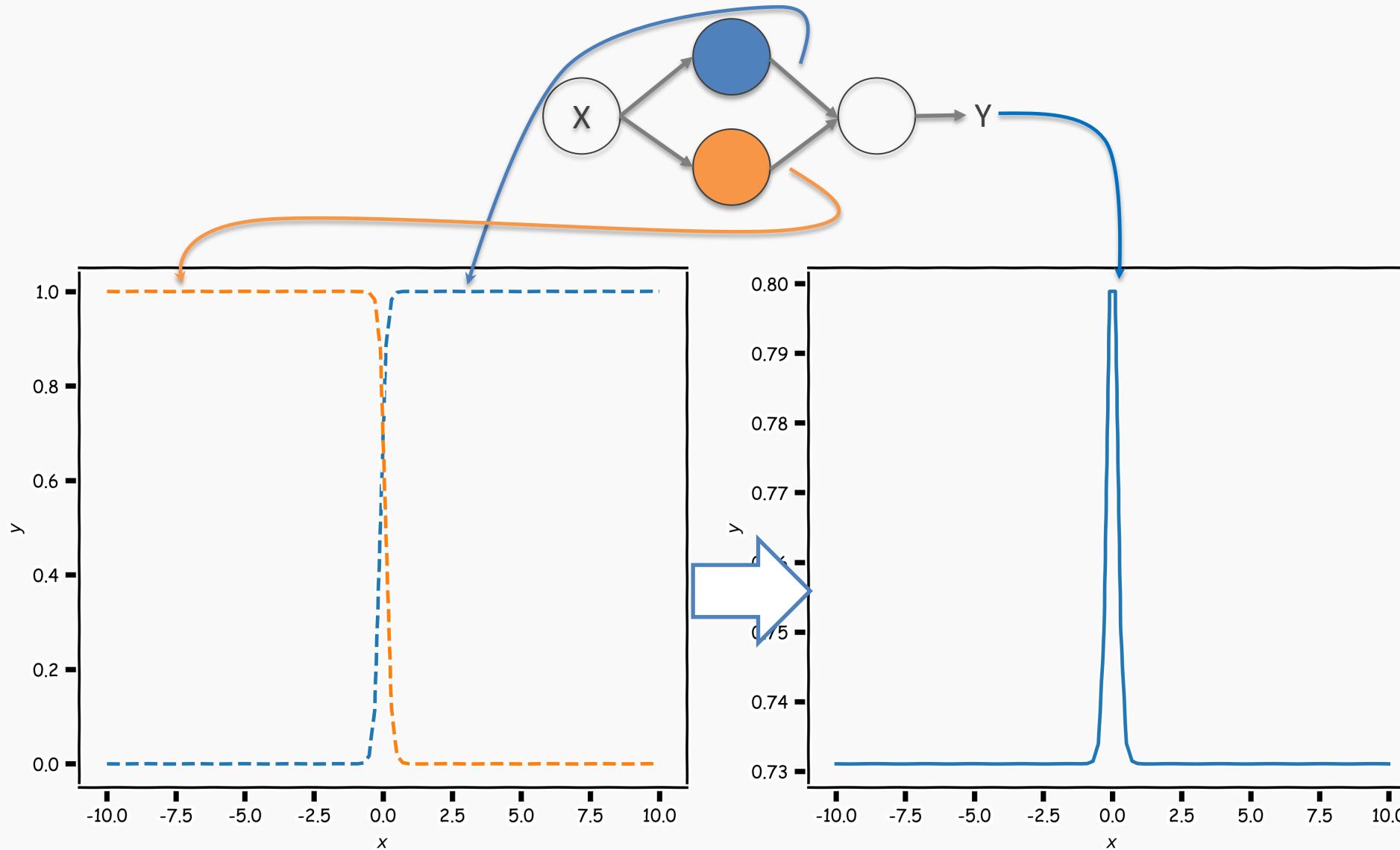


Single Neuron Network
Very similar to Perceptron

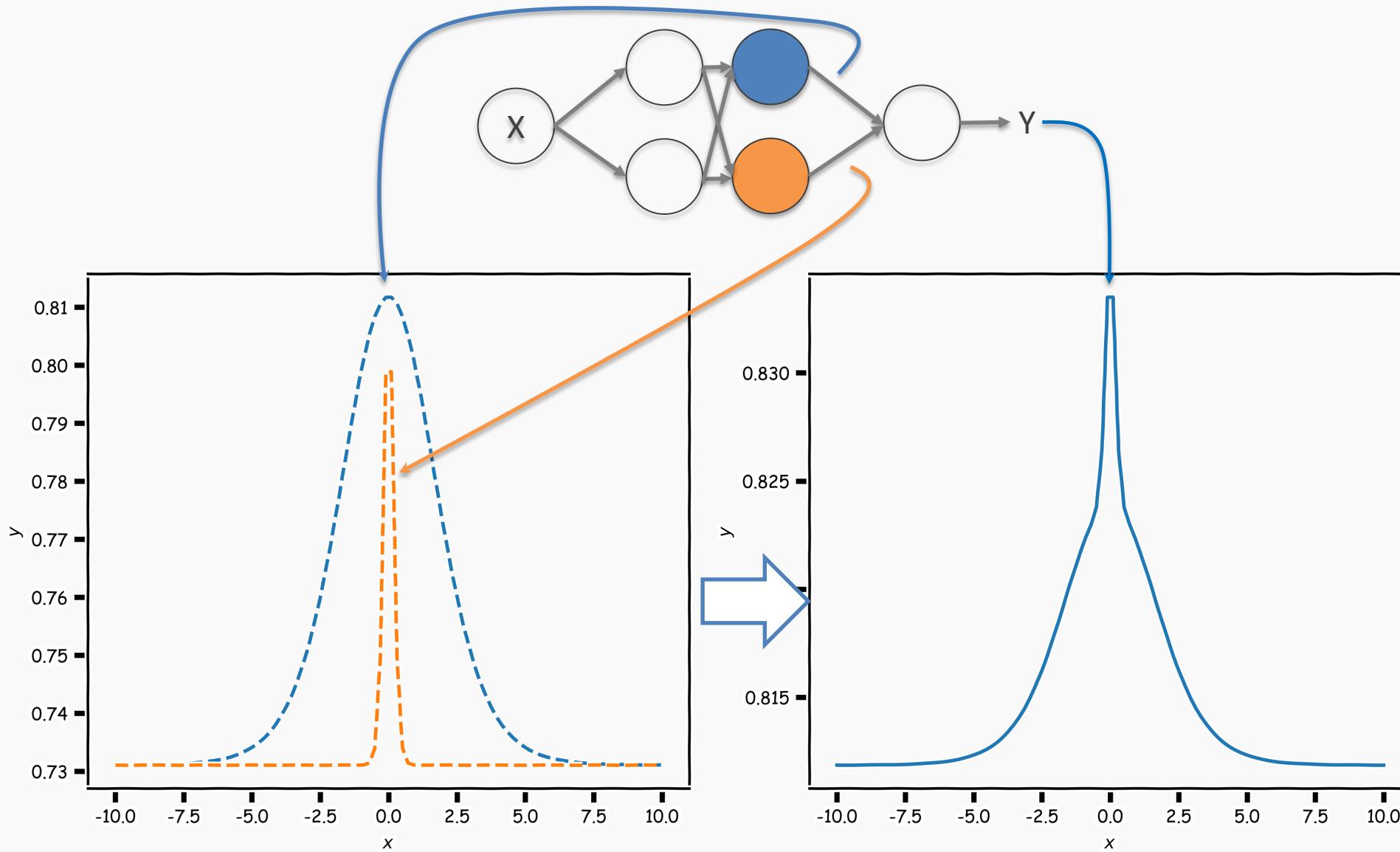
Combining neurons allows us to model interesting functions



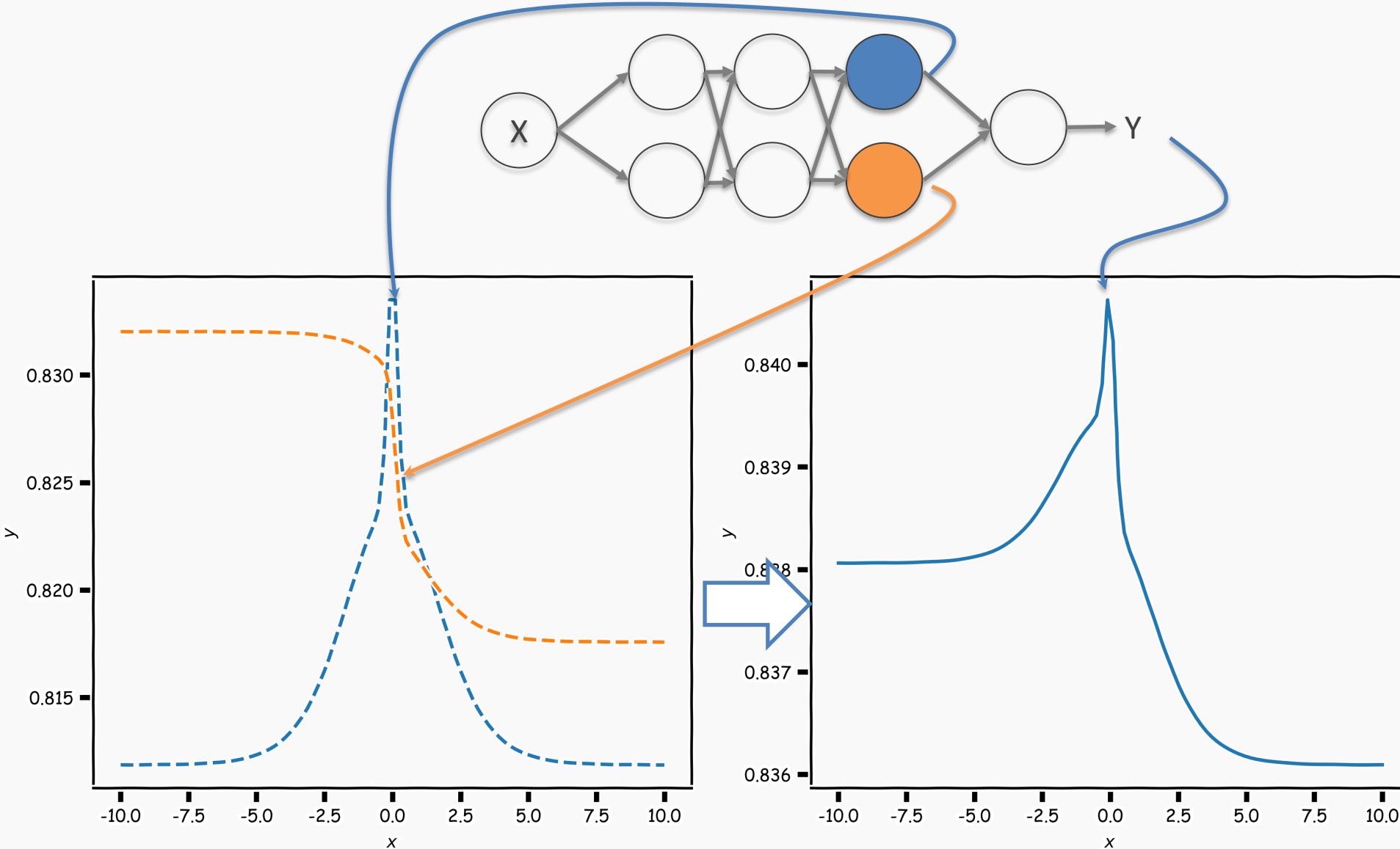
Different weights change the shape and position



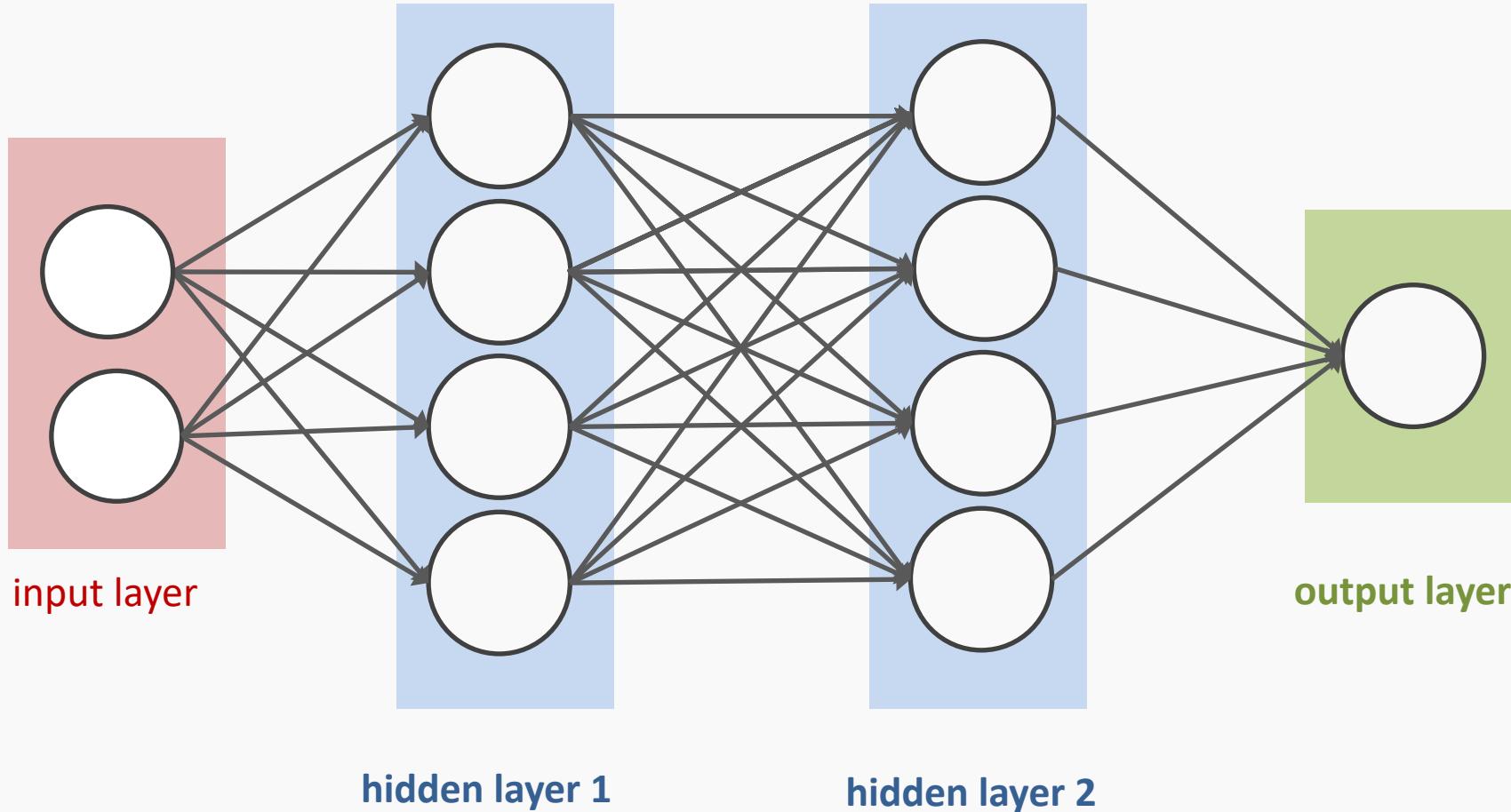
Neural networks can model *any* reasonable function



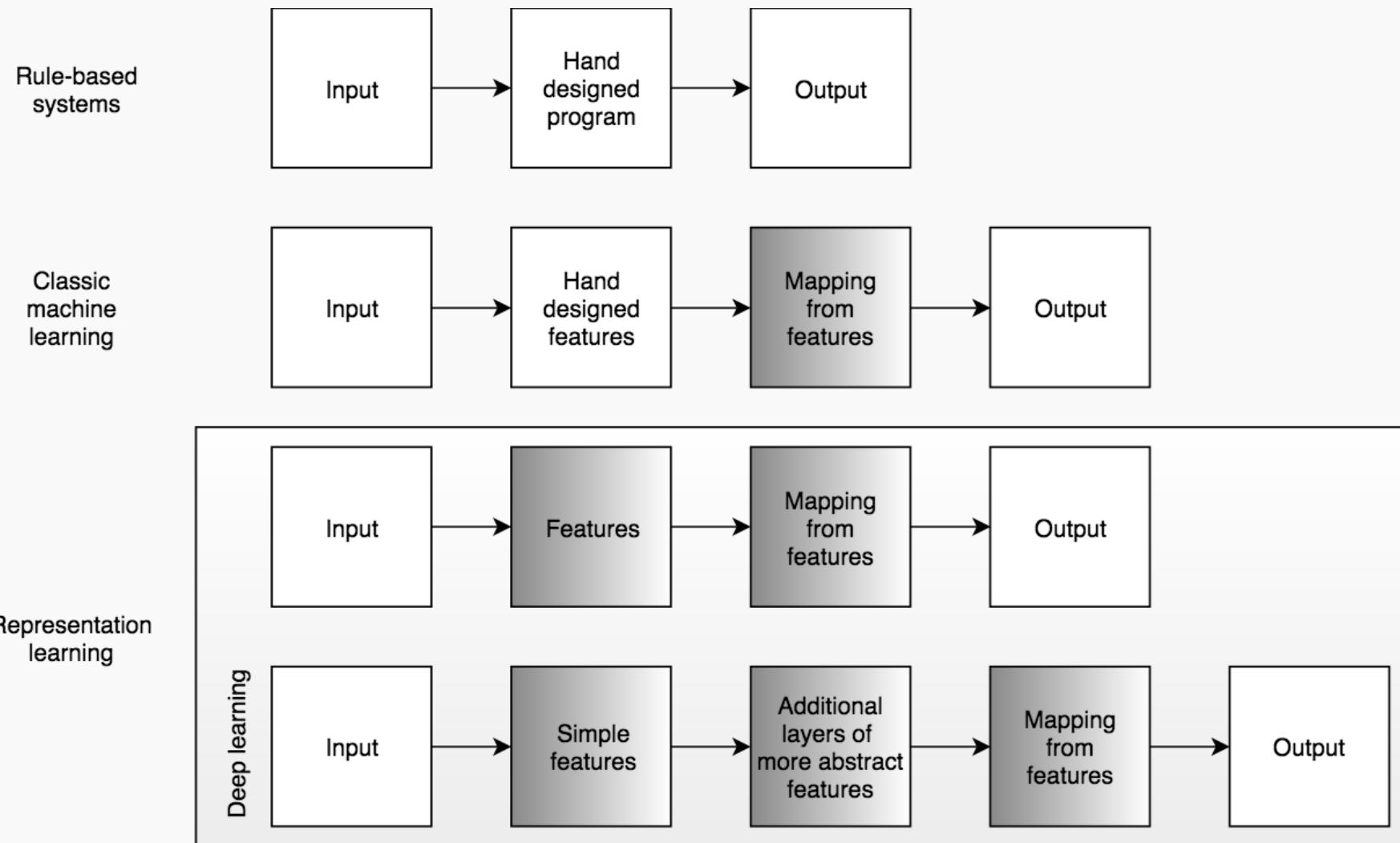
Adding layers allows us to model increasingly complex functions



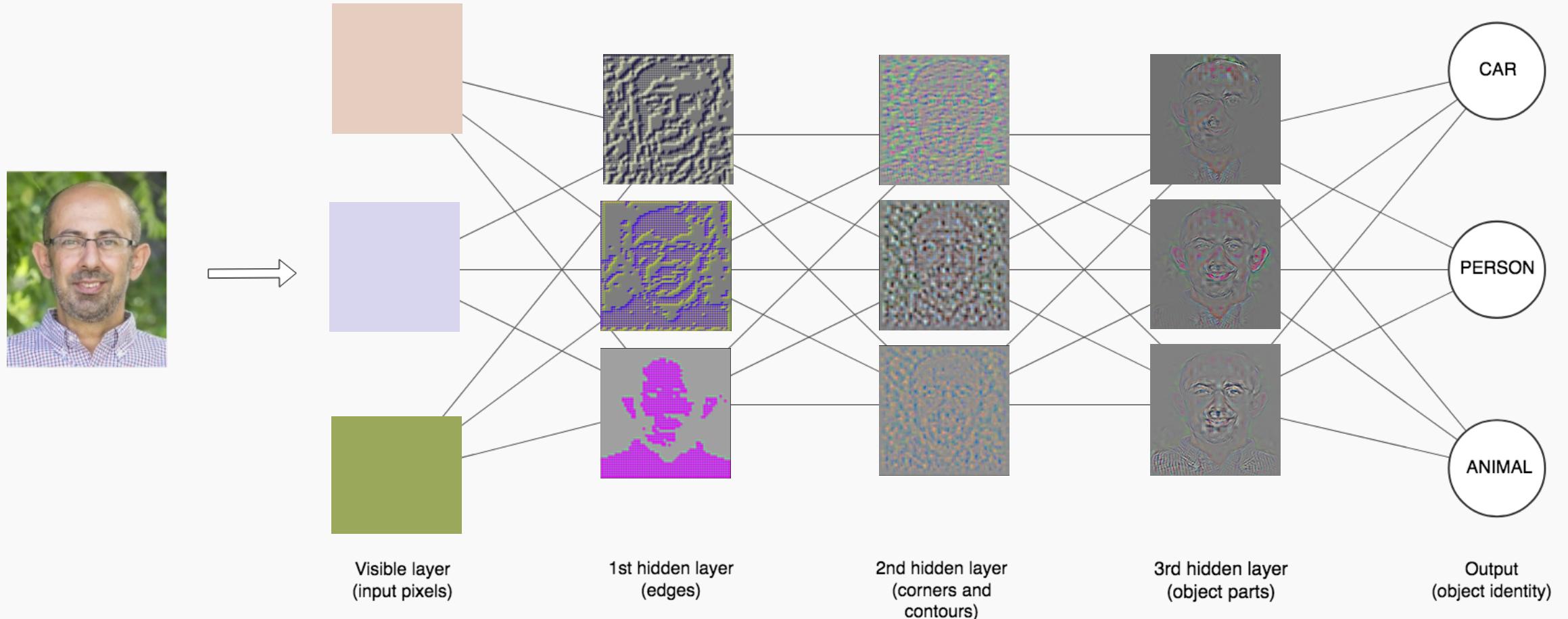
Anatomy of artificial neural network (ANN)



Learning Multiple Components

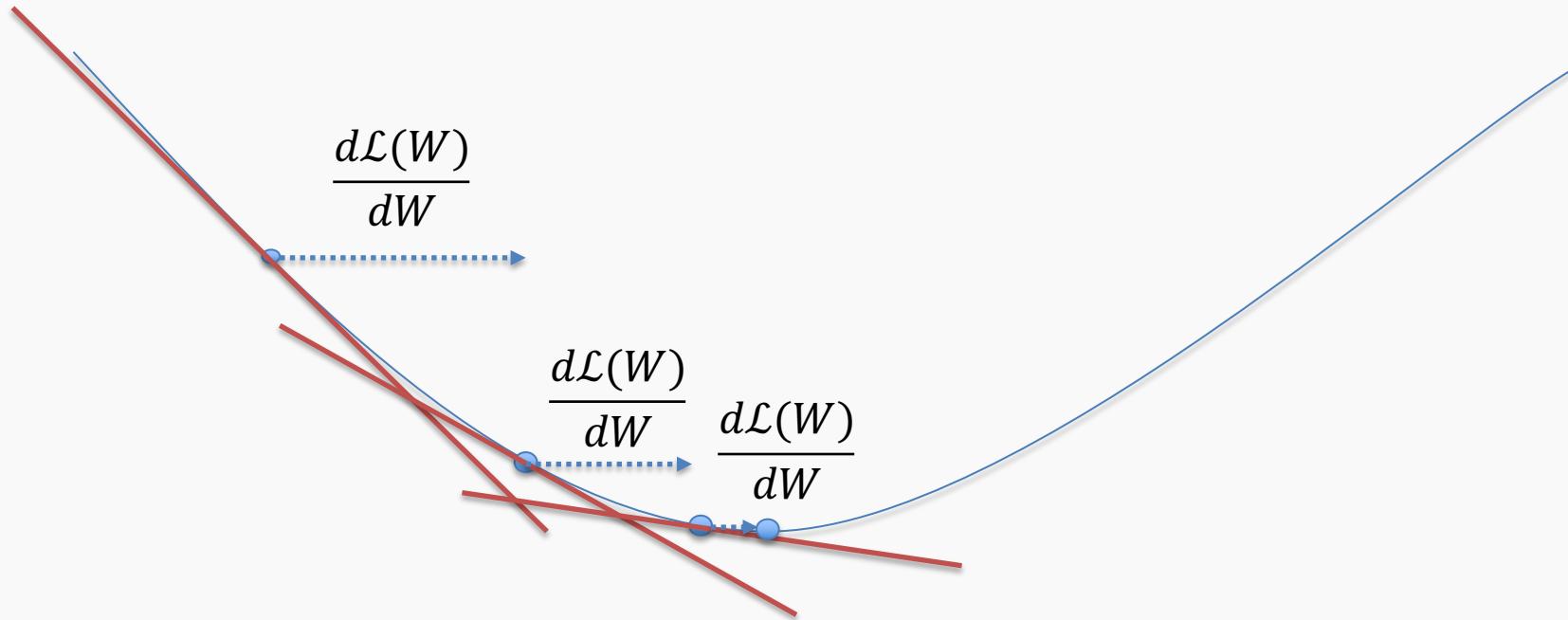


Depth = Repeated Compositions



Gradient Descent (cont.)

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

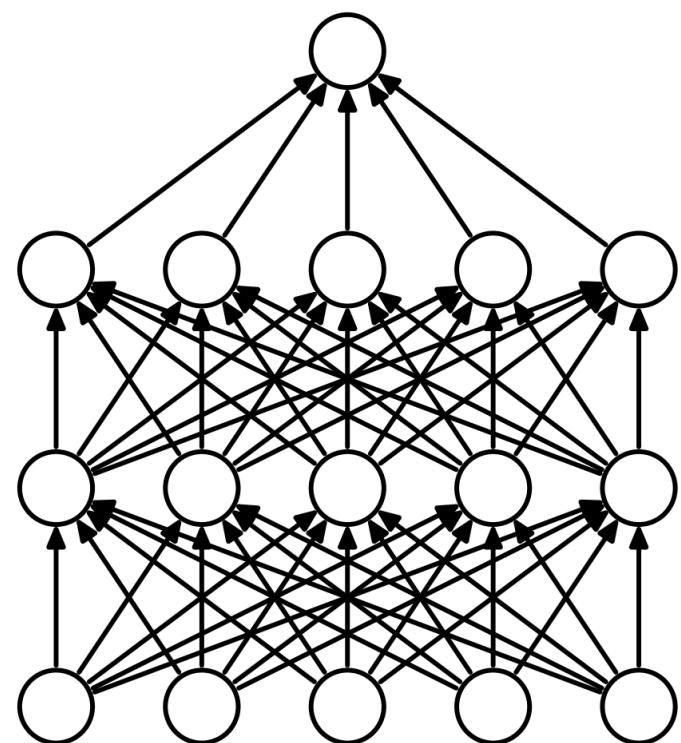


Data Augmentation

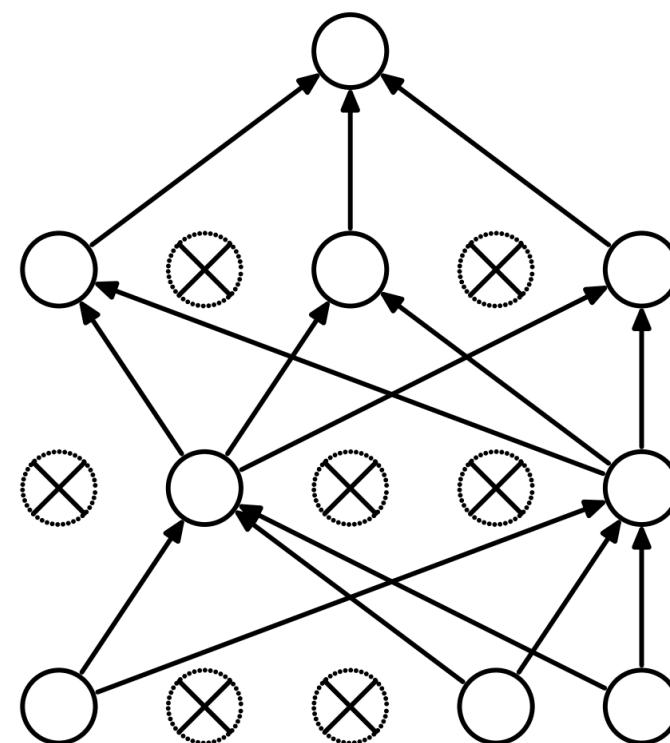


Dropout

- 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



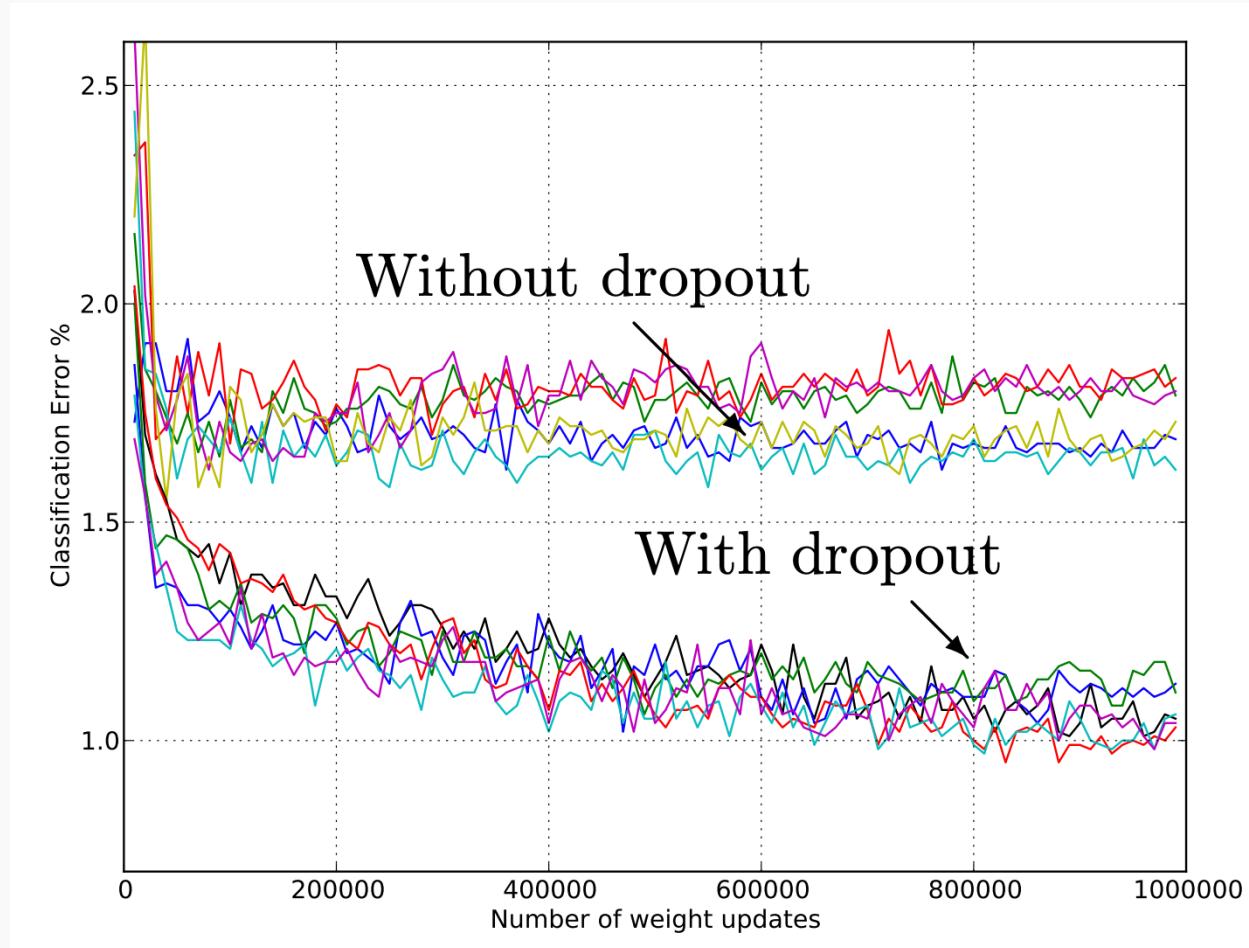
(a) Standard Neural Net



(b) After applying dropout.

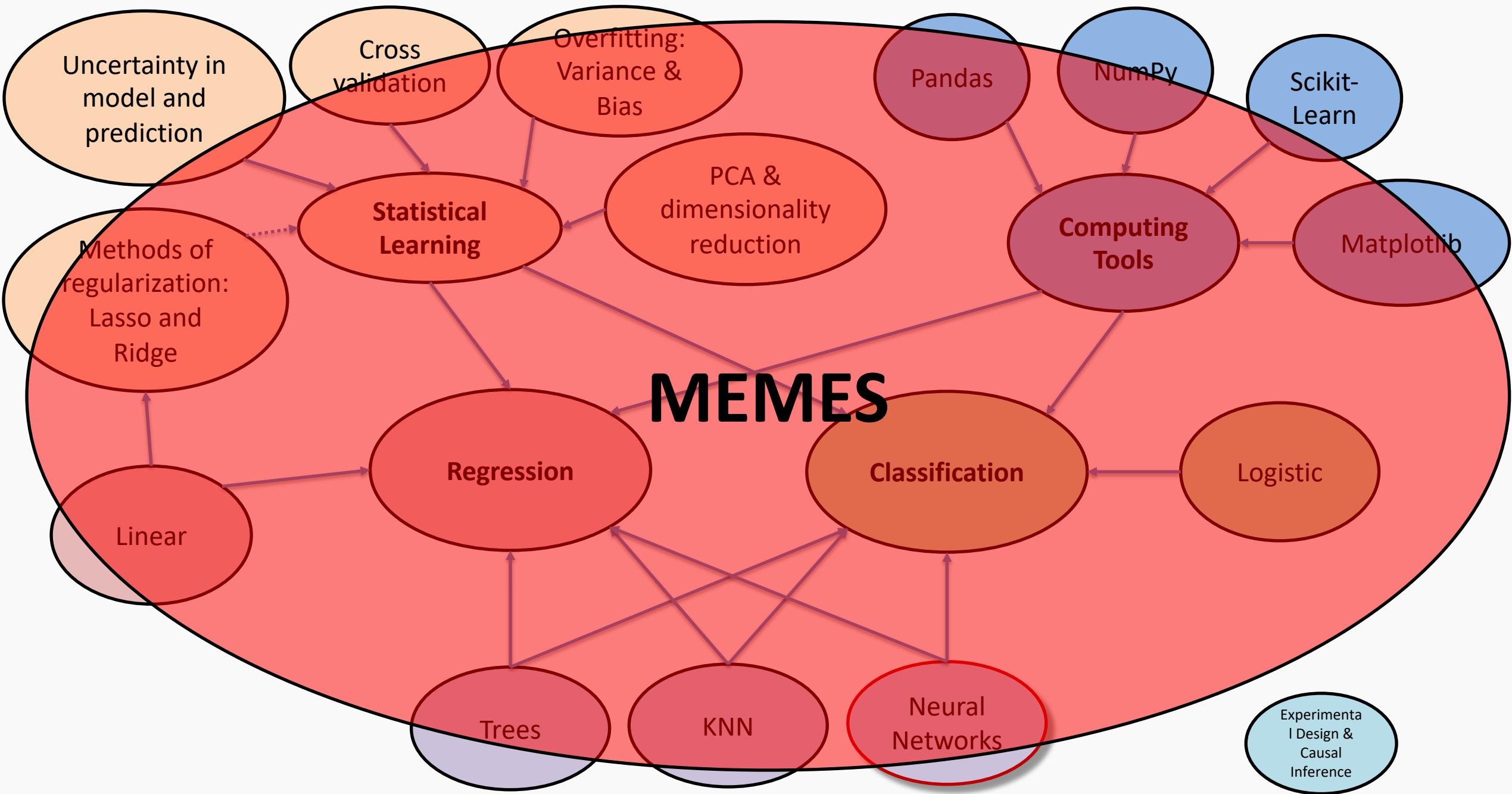
Dropout

- 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

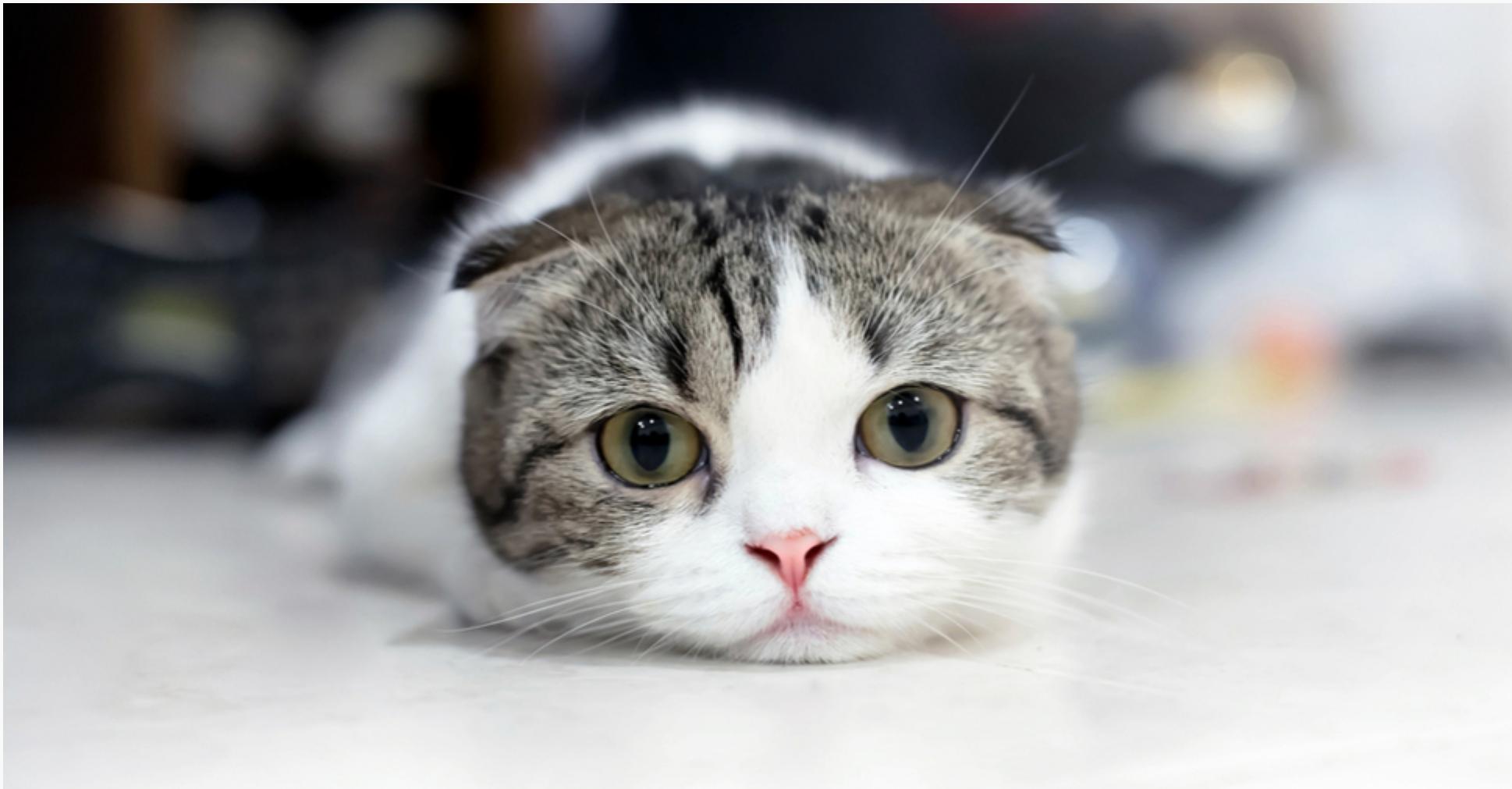
When you're a kNN with $k = 2$



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



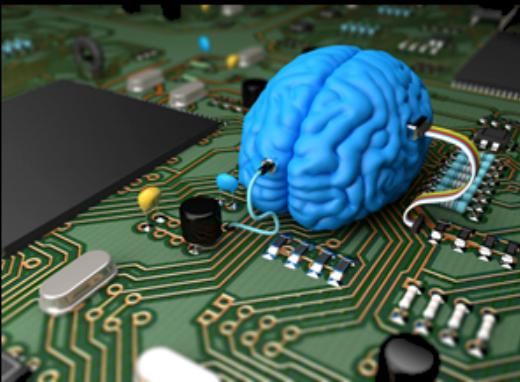
PPppBoost



Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do



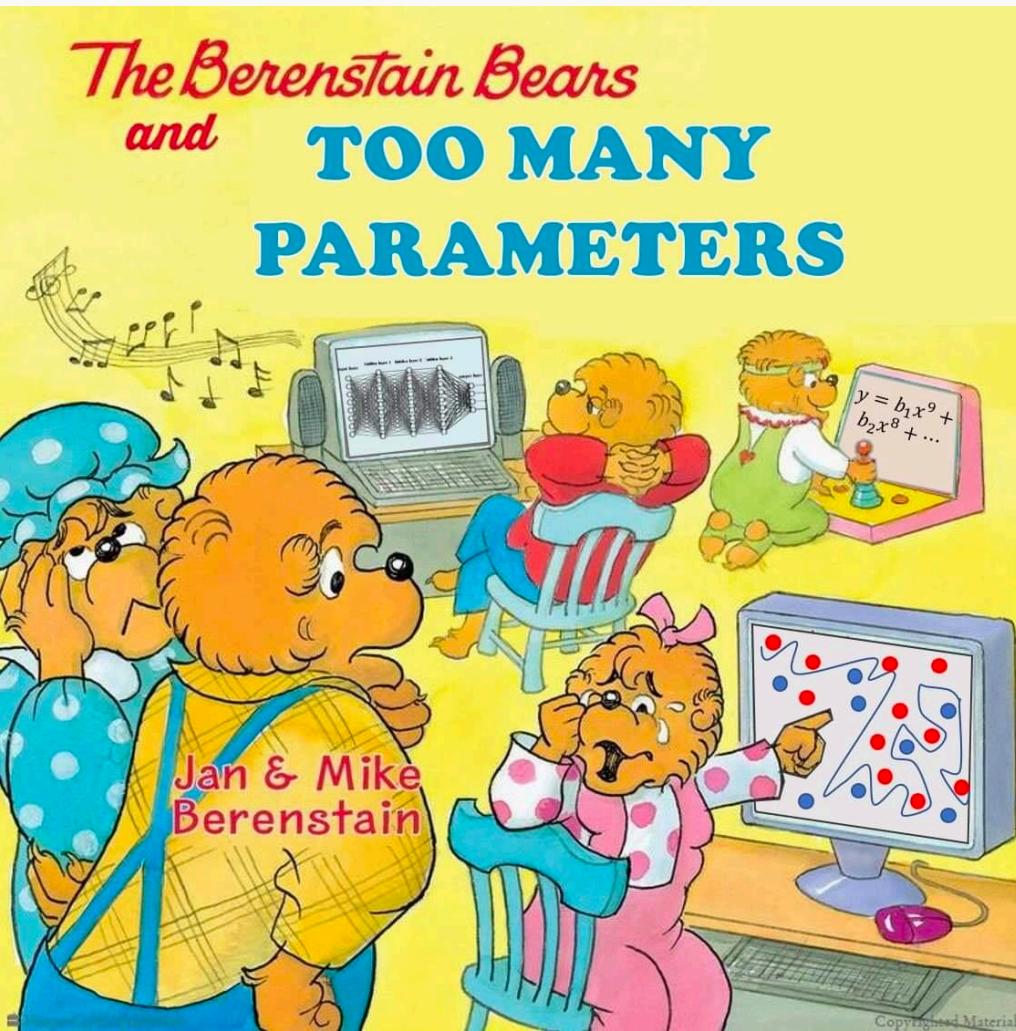
What mathematicians think I do



What I think I do

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

What I actually do



Better than AB...



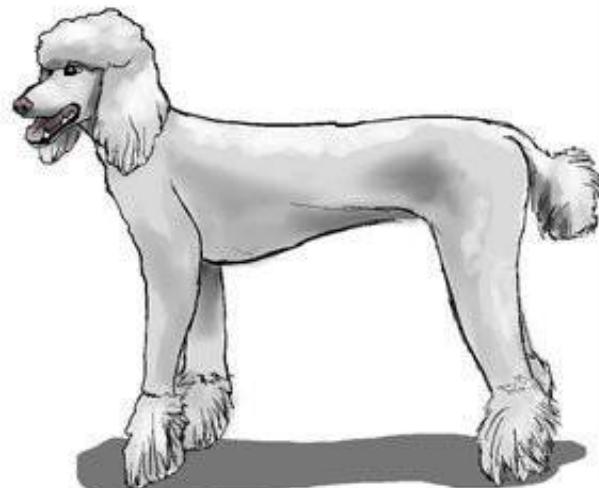
Scalar



Vector



Matrix

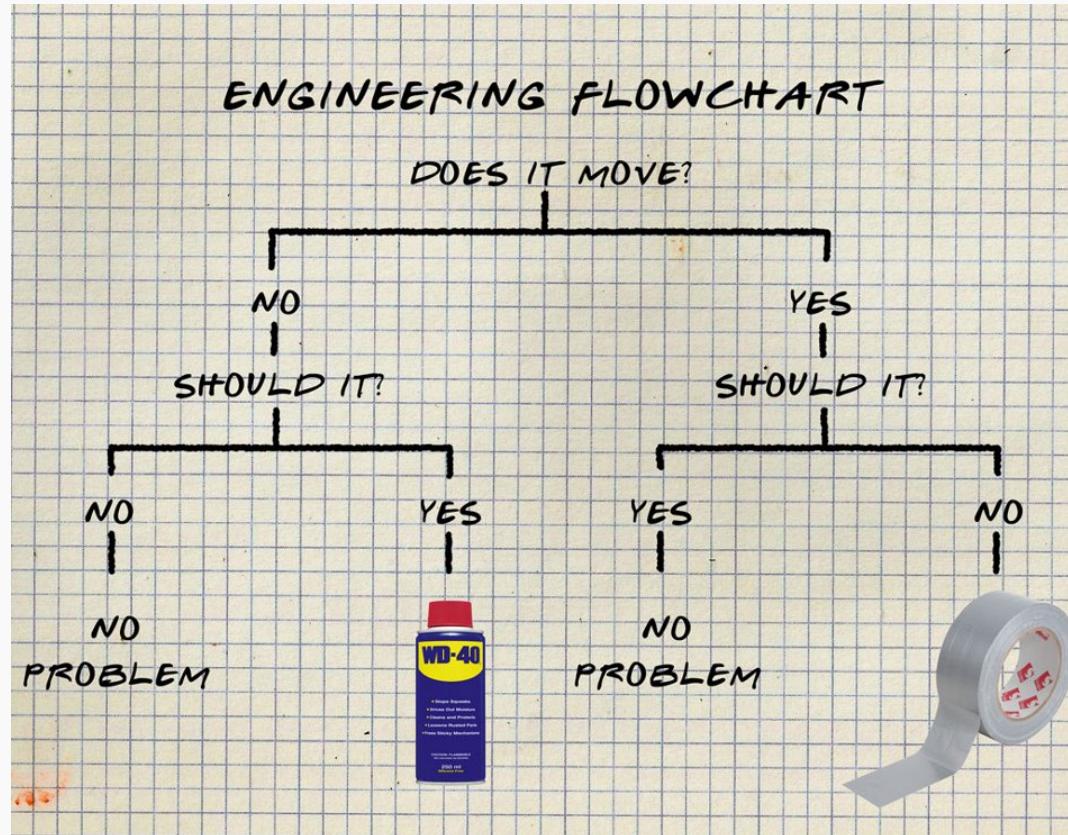


Tensor

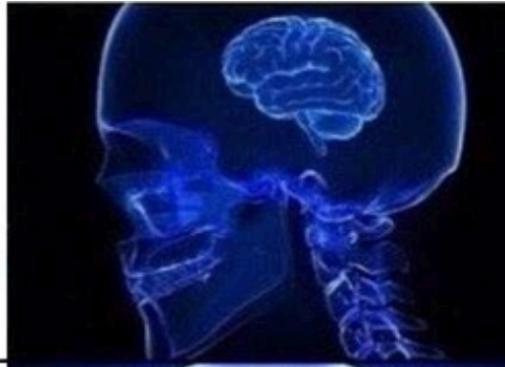


Interpretable Models

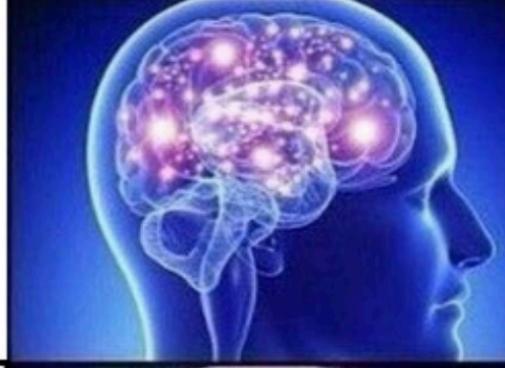
People in every walk of life have long been using interpretable models for differentiating between classes of objects and phenomena:



**INCLUDING
DROPOUT LAYERS**



**SCHEDULING
THE
LEARNING RATE**



**EXPERIMENTING
WITH
ACTIVATION FUNCTIONS**



**OPTIMIZING
THE RANDOM SEED**

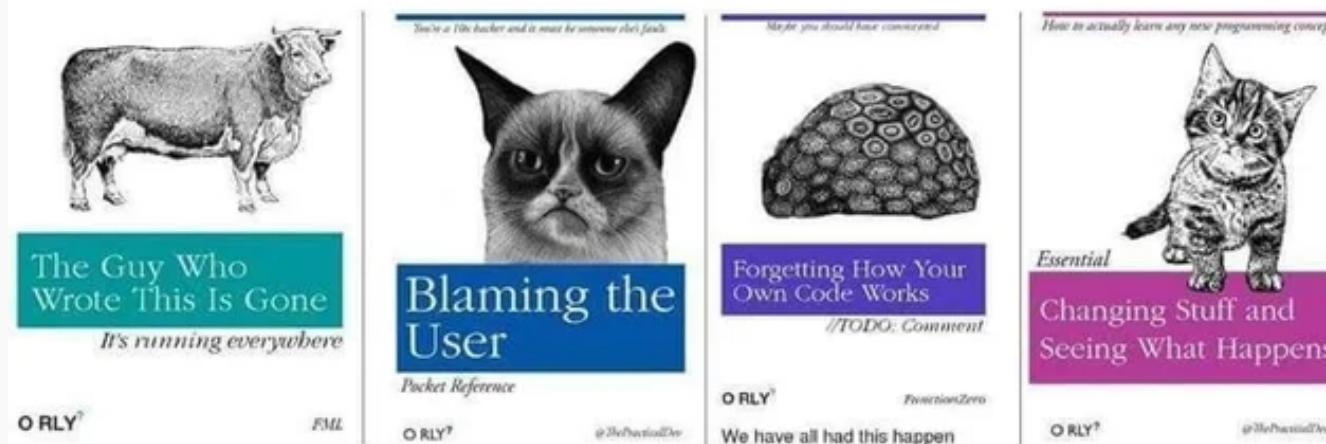
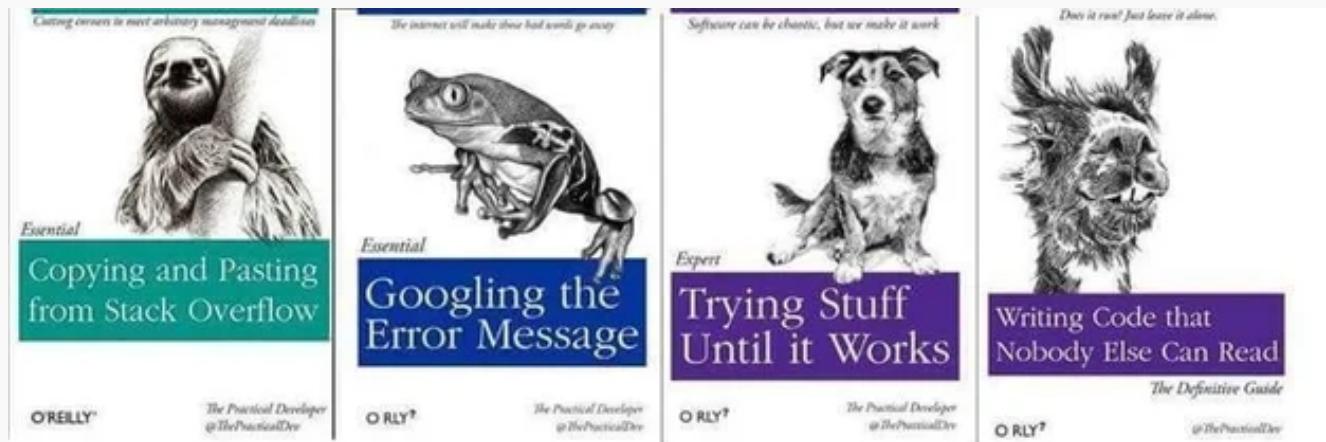


BOOTSTRAPPING

I DONT THINK IT MEANS WHAT YOU
THINK IT MEANS

memegenerator.net





ANNOUNCEMENTS

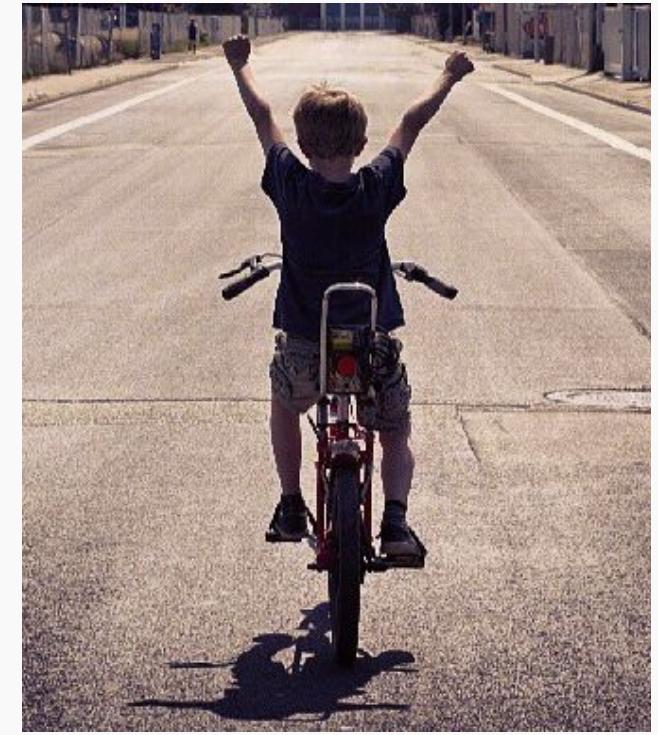
- Homework 7 individual



HW1



HW4



HW7

Without OH or ED forum

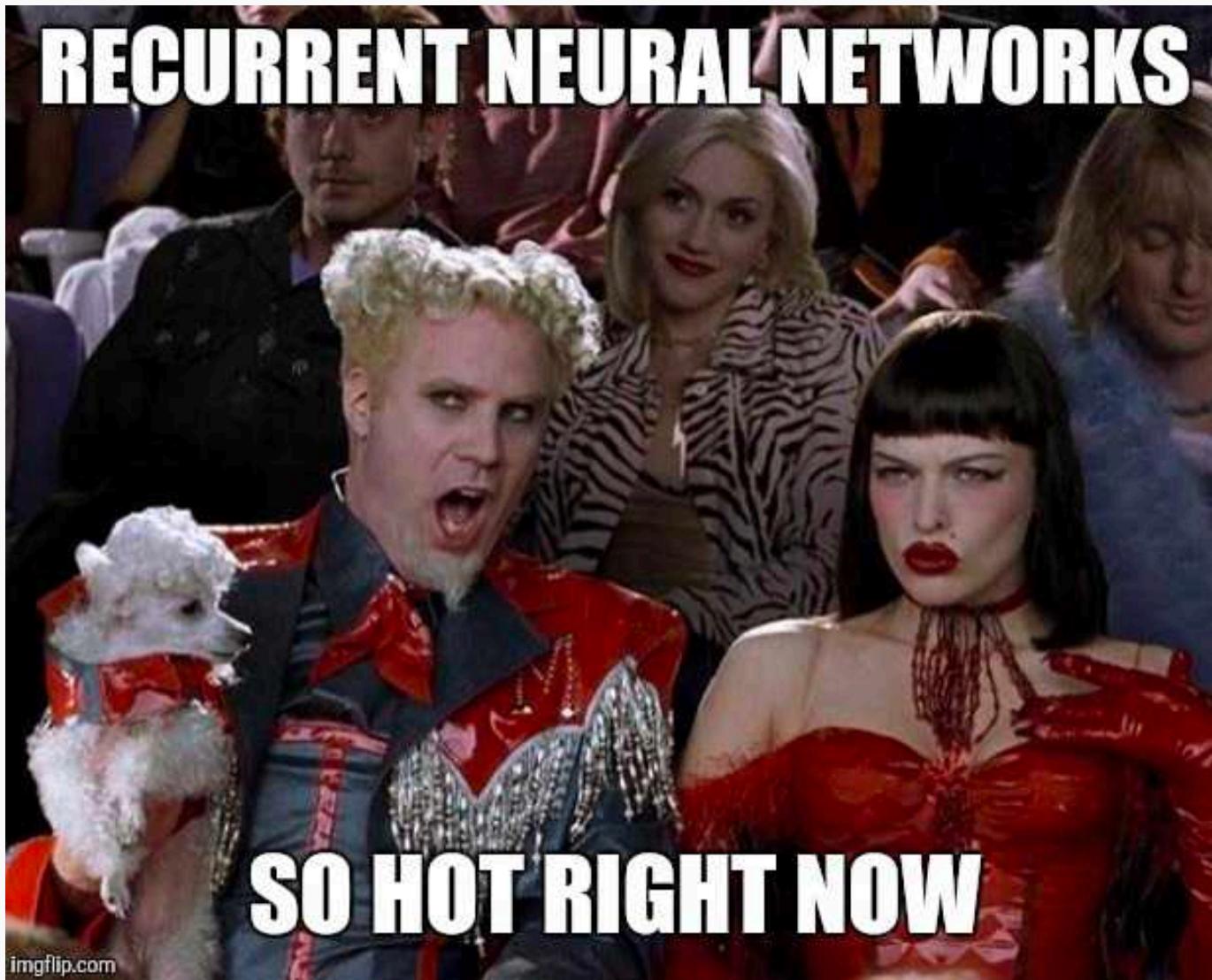
ANNOUNCEMENTS

- 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?



CS109B

Instructors: Pavlos Protopapas, Mark Glickman, Chris Tanner

Topics

HW: Similar to 109A

Modules

Medical School: Mauricio Santillana

FAS: Doug Finkbeiner

GSD: Garcia del Castillo Lopez

HCSPH: Francesca Dominici



Thank You!

