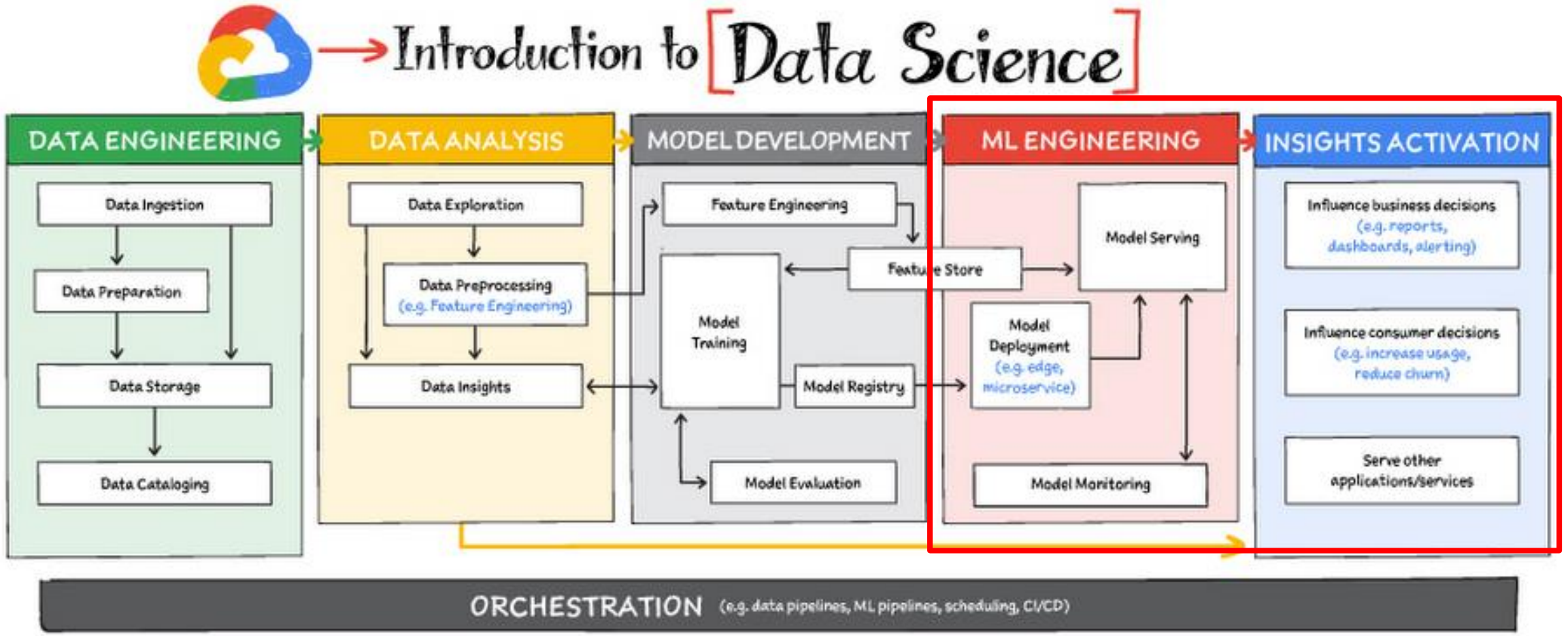
# Explainable AI

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

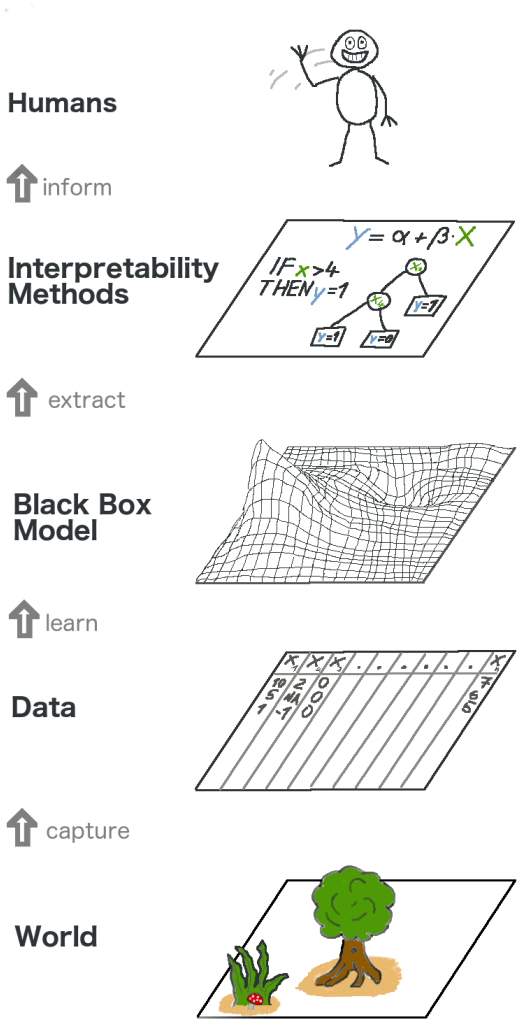
# The Pipeline




https://cloud.google.com/blog/topics/developers-practitioners/intro-data-science-google-cloud

# A recap of the bigger picture

- It's great that *interpretable machine learning* fuses the work of *statisticians* and *machine learning* specialists
  - Traditionally, statisticians deal with the Data layer, such as planning clinical trials or designing surveys. They skip the Black Box Model layer and go right to the Interpretability Methods layer
  - Machine learning specialists also deal with the Data layer, such as collecting labeled samples of skin cancer images or crawling Wikipedia. Then they train a black box model. The Interpretability Methods layer is often skipped and humans directly deal with the black box model predictions!




https://christophm.github.io/interpretable-ml-book/agnostic.html

# Why interpretable model?

- The importance of interpretability
  1. **Reliability -** A self-driving car automatically detects cyclists based on a deep learning system. You want to be 100% sure that the abstraction the system has learned is error-free, because running over cyclists is quite bad
  2. **Fairness -** A model says a bank shouldn't loan someone money, and the bank is legally required to explain the basis for each loan rejection
  3. **Trust -** The process of integrating machines and algorithms into our daily lives requires interpretability to increase social acceptance
  4. **Causality -** A healthcare provider wants to identify what factors are driving each patient's risk of disease so they can directly address those factors with targeted health interventions
- Models can only be *debugged and audited* when they can be interpreted
  - You can then directing future data collection or informing human decision-making, etc

# What types of insights are possible

- Many people say machine learning models are "black boxes", in the sense that they can make good predictions but you can't understand the logic
  - This statement is true in the sense that some data scientists don't know how to extract insights from models yet
- However, there are techniques to extract the following insights
  - How does the learning algorithm create the model?
  - How does the trained model make predictions?
  - What features in the data did the model think are most important? How does each feature affect the model's predictions in a big-picture sense?
  - For any single prediction from a model (explanation), how did each feature in the data affect that particular prediction?

# What is a good explanation for human?

1. Explanations are *contrastive* - The best explanation is the one that *highlights the greatest difference* between the object of interest and the reference object

- x A complete explanation why the drug does not work might include: The patient has had the disease for 10 years, 11 genes are over-expressed, the patients body is very quick in breaking the drug down into ineffective chemicals, …
  - ✓ A contrastive explanation might be much simpler: In contrast to the responding patient, the non-responding patient has a certain combination of genes that make the drug less effective
- Creating contrastive explanations is application-dependent and solutions for the automated creation of contrastive explanations might also involve finding *prototypes* in the data

2. Explanations are *selected* - People do not expect explanations that cover the actual and complete list of causes of an event

- ✓ Make the explanation very short, give only 1 to 3 reasons, even if the world is complex

# What is a good explanation for human?

3. Explanations *focus on the abnormal* - If one of the input features for a prediction was abnormal in any sense (like a rare category of a categorical feature) and influenced the prediction, it should be included in an explanation

- An abnormal feature in our house price prediction example might be that a rather expensive house has three balconies

4. Good explanations are consistent with *prior beliefs of the explainee* - Humans tend to ignore information that is inconsistent with their prior beliefs

5. Good explanations are *general and probable* - A cause that can explain many events is very general and could be considered a good explanation

# 1. Interpretable Models (intrinsically interpretable models)

- Use models that are intrinsically interpretable if possible
  - A model with monotonicity constraints ensures that the relationship between a feature and the target outcome always goes in the same direction over the entire range of the feature
    - Monotonicity is useful for the interpretation of a model because it makes it easier to understand a relationship

| Algorithm | Linear | Monotone | Interaction | Task |
|---|---|---|---|---|
| k-nearest neighbors | No | No | No | class, regr |
| Linear regression | Yes | Yes | No | regr |
| Logistic regression | No | Yes | No | class |
| Naive Bayes | No | Yes | No | class |
| Decision trees | No | Some | Yes | class, regr |

# Interpretable Models - Decision Rules

- Decision rules follow a general structure: *IF* the conditions are met *THEN* make a certain prediction
  - New in machine learning is that the decision rules are *learned* through an algorithm
- The baseline `OneR` algorithm selects the one that carries the most information about the outcome of interest and creates decision rules from this feature
  1. Discretize the continuous features by choosing appropriate intervals
  2. For each feature:
     1. Create a cross table between the feature values and the (categorical) outcome
     2. For each value of the feature, create a rule which predicts the *most frequent* class of the instances that have this particular feature value
     3. Calculate the total error of the rules for the feature
  3. Select the feature with the smallest total error

| Location | Size | Pets | Price |
|---|---|---|---|
| good | small | yes | high |
| good | big | no | high |
| good | big | no | high |
| bad | medium | no | medium |
| good | medium | only cats | medium |
| good | small | only cats | medium |
| bad | medium | yes | medium |
| bad | small | yes | low |
| bad | medium | yes | low |
| bad | small | no | low |

| | Price=low | Price=medium | Price=high |
|---|---|---|---|
| Location=bad | 3 | 2 | 0 |
| Location=good | 0 | 2 | 3 |

$\frac{4}{10}$

| | Price=low | Price=medium | Price=high |
|---|---|---|---|
| Size=big | 0 | 0 | 2 |
| Size=medium | 1 | 3 | 0 |
| Size=small | 2 | 1 | 1 |

$\frac{3}{10}$

| | Price=low | Price=medium | Price=high |
|---|---|---|---|
| Pets=no | 1 | 1 | 2 |
| Pets=only cats | 0 | 2 | 0 |
| Pets=yes | 2 | 1 | 1 |

$\frac{4}{10}$

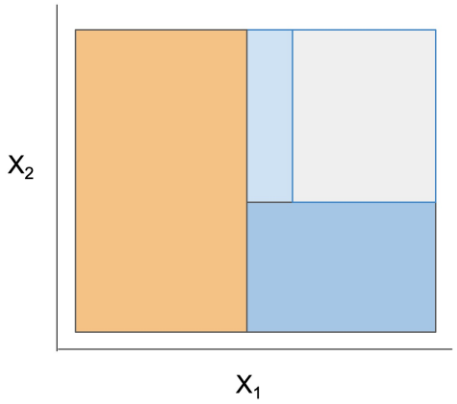
IF Size=small THEN Price=low
ELSE IF Size=medium THEN Price=medium
ELSE IF Size=big THEN Price=high

# Interpretable Models - Decision Rules

- OneR prefers features with *many possible levels* like decision tree
  - Imagine a dataset that contains only noise and no signal. Some features have more levels than others. The features with more levels can now more easily overfit. A feature that has a separate level for each instance from the data would perfectly predict the entire dataset
- There are many other popular alternatives: sequential covering, Bayesian rule lists, Rulefit, FIGS

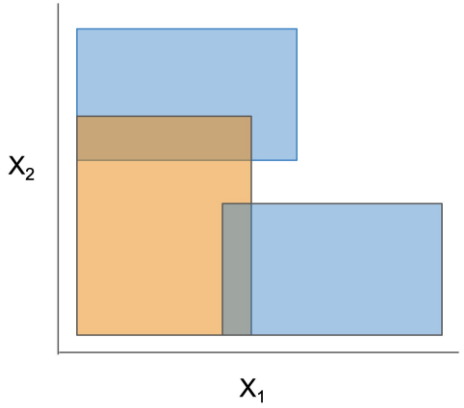
Rule list

**IF** $X_1 < 5$: p(+) = 0.3
**ELSE** If $X_2 < 4$: p(+) = 0.9
**ELSE** If $X_1 > 6$: p(+) = 0.7
**ELSE** p(+) = 0.5

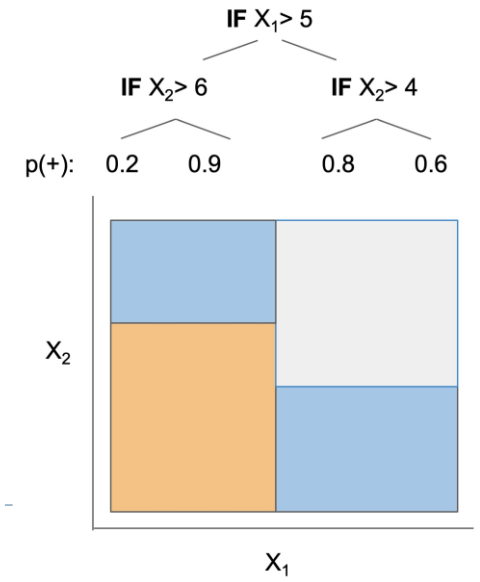



Rule set

**IF** $X_1 < 6$ and $X_2 > 6$: p(+) = 0.8
**IF** $X_1 < 5$ and $X_2 < 7$: p(+) = 0.3
**IF** $X_1 > 4$ and $X_2 < 4$: p(+) = 0.8




Rule tree



# Interpretable Models - Explainable Boosting Machine (EBM)

- EBM is a generalized additive model (GAM). The GAM can be written as

$$y_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i$$

It is called an additive model because we calculate a separate $f_j$ for each $X_j$, and then add together all of their contributions

- EBM learns each feature function $f_j$ using modern machine learning techniques such as bagging and gradient boosting in round-robin fashion
- EBM can automatically detect and include *pairwise interaction terms* of the form

$$y_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \sum_{i,j} f_{ij}(x_i, x_j) + \epsilon_i$$

# Model-Agnostic Methods (post hoc interpretation methods)

- The great advantage of *model-agnostic* interpretation methods over model-specific ones is their flexibility
  - **Model flexibility**: The interpretation method can work with any machine learning model, such as random forests and deep neural networks
  - **Explanation flexibility**: You are not limited to a form of explanation. In some cases it might be useful to have a dependence plot, in other cases a graphic with feature importance
  - **Representation flexibility**: The explanation system should be able to use different feature representations as the model being explained. For a text classifier uses word embedding vectors, it might be preferable to use the presence of individual words for the explanation
- Model-agnostic interpretation methods can be further distinguished into *local* and *global* methods
  - Global methods describe how features *affect the prediction on average*. In contrast, local methods aim to *explain individual predictions*

# 2. Global Methods - Partial dependence plots (PDP)

- While feature importance shows what variables most affect predictions, partial dependence plots show *how a feature affects predictions*
- This is useful to answer questions like:
  - Controlling for all other house features, what impact do longitude and latitude have on home prices?
- If you are familiar with linear or logistic regression models, partial dependence plots can be interpreted similarly to the *coefficients* in those models!
  - Though, partial dependence plots on **sophisticated models** can capture more complex patterns than coefficients from simple models. Like permutation importance, partial dependence plots are calculated after a *model has been fitted*

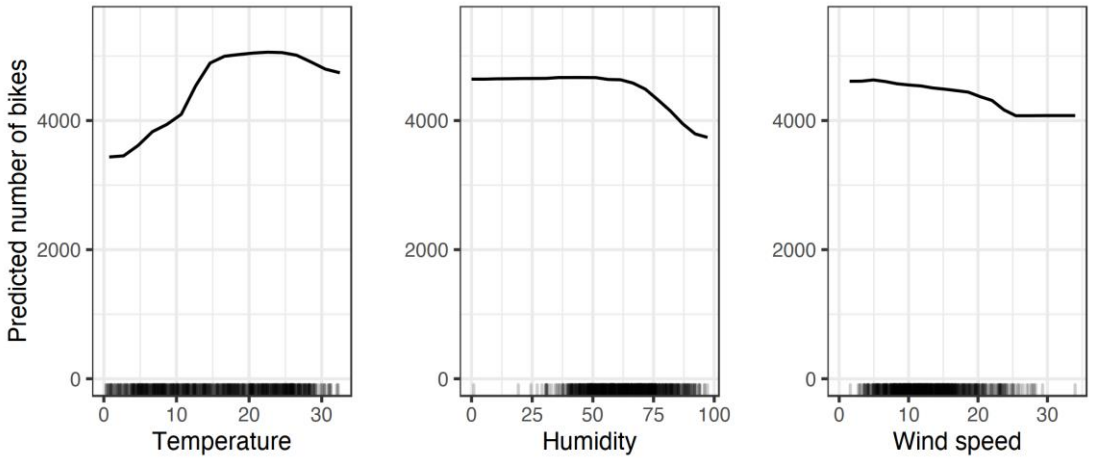
# Partial dependence plots (PDP)

- The partial dependence function for regression is defined as

$$\hat{f}_S(x_S) = E_{X_C}\left[\hat{f}(x_S, X_C)\right] = \int \hat{f}(x_s, X_C)p(x_C)dx_C = \frac{1}{n}\sum_{i=1}^{n} \hat{f}\left(x_S, X_C^{(i)}\right)$$

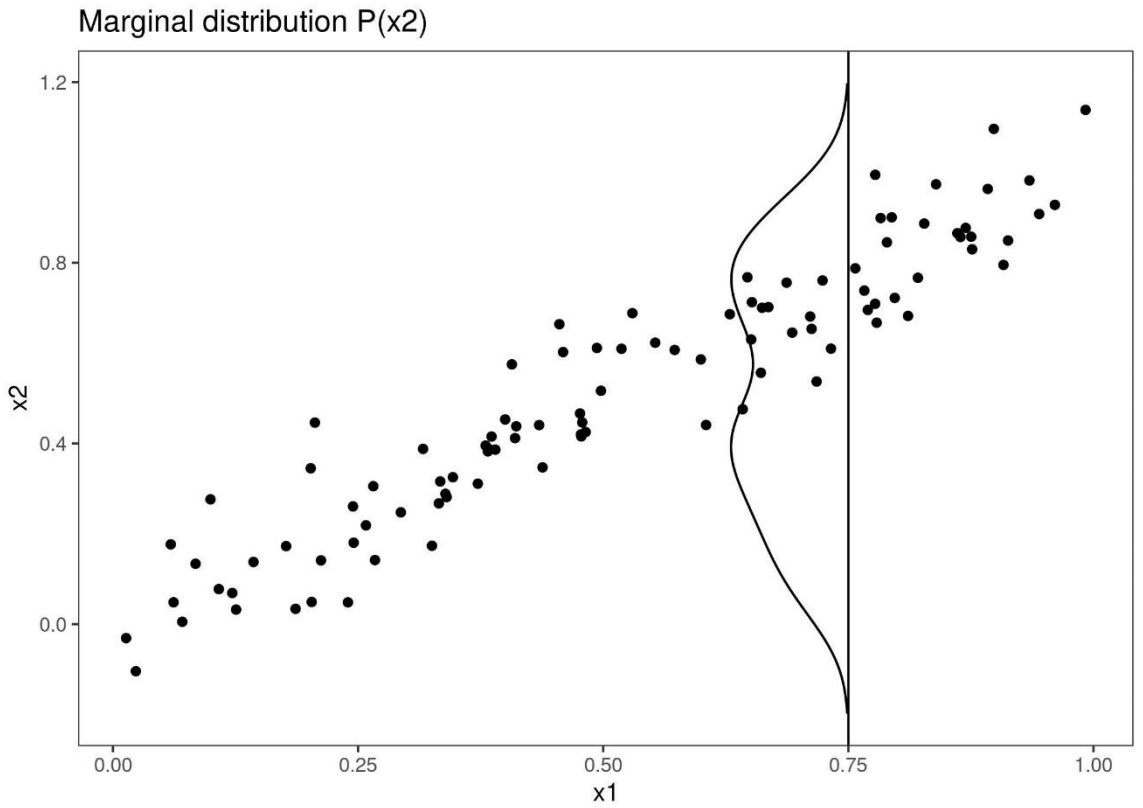
The $x_S$ are the features for which the partial dependence function should be plotted and $X_C$ are the other features used in the model $\hat{f}$, which are here treated as random variables

- It works by marginalizing the model output over the distribution of the features in set $C$, so that the function shows the relationship between the $S$ we are interested in and the predicted outcome. Computing this integral for various values of $x_s$ produces a *PDP plot*
- It is assumed that the features in $C$ are not correlated with the features in $S$



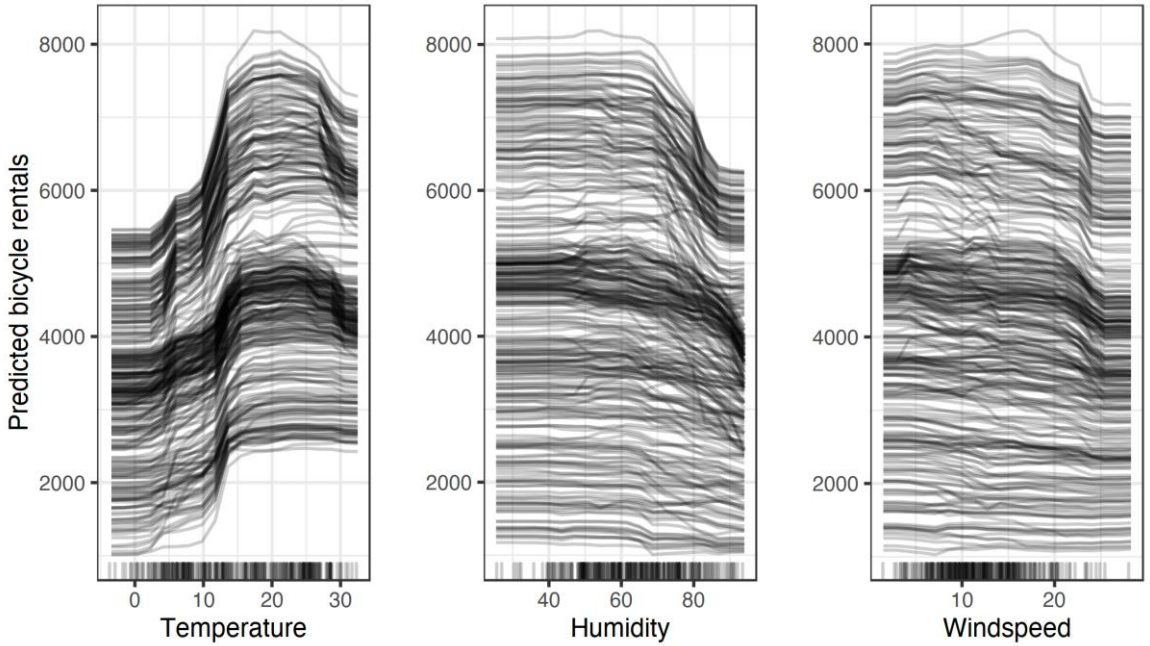
# Partial dependence plots (PDP)

- Strongly correlated features $X_1$ and $X_2$. To calculate the feature effect of $X_1$ at 0.75, the PDP replaces $X_1$ of all instances with 0.75, falsely assuming that the distribution of $X_2$ at $X_1 = 0.75$ is the same as the marginal distribution of $X_2$!
  - This results in unlikely combinations of $X_1$ and $X_2$ (e.g. $X_2 = 0.2$ at $X_1 = 0.75$ ), which the PDP uses for the calculation of the average effect
  - In this case Accumulated Local Effects (ALE) Plot may be better suited



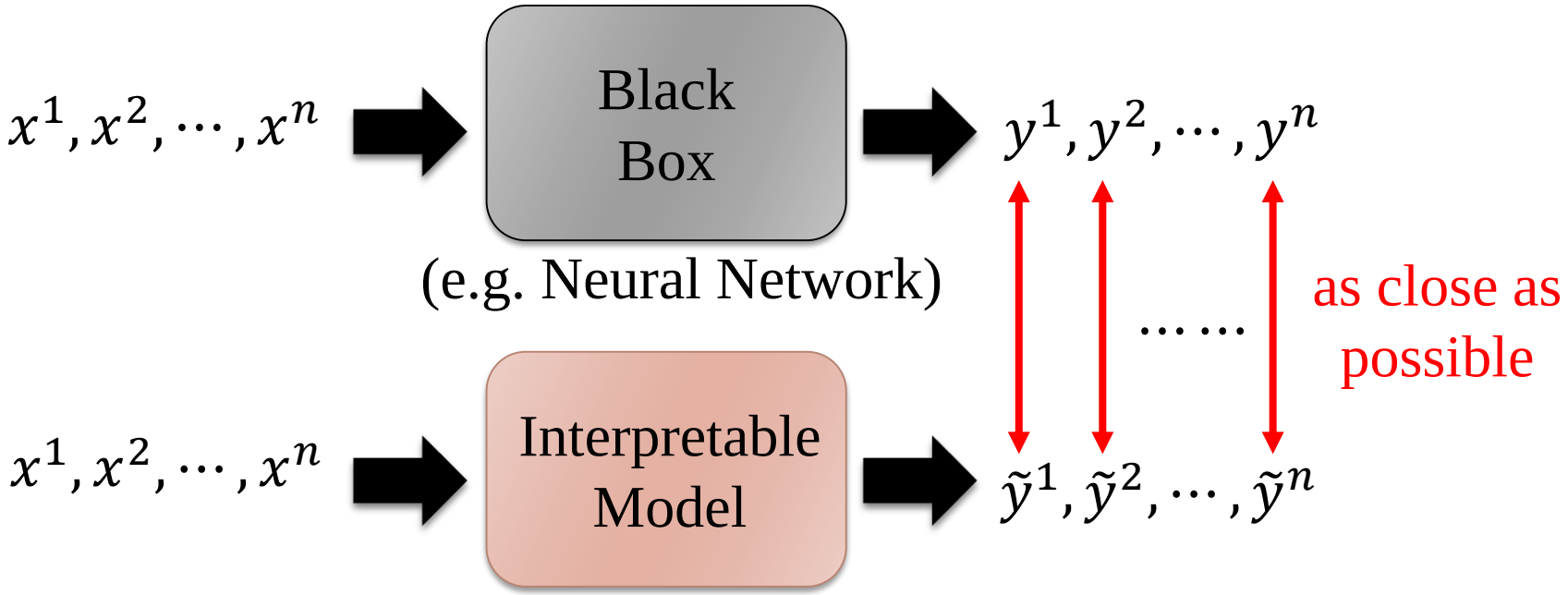
# 3. Local Methods - Individual conditional expectation (ICE) plot

- Similar to a PDP, an individual conditional expectation (ICE) plot shows the dependence between the target function and an input feature of interest
  - Unlike a PDP, which shows the *average* effect of the input feature, an ICE plot visualizes the dependence of the prediction *on a feature for each sample* separately with one line per sample. A PDP is the average of the lines of an ICE plot
  - In ICE plots, for each instance in $\{(x_S, X_C^{(i)})\}_{i=1}^{n}$ the curve $\hat{f}_S^{(i)}$ is plotted against $x_S$
  - Unlike PDP, ICE curves can uncover heterogeneous relationships



# Local Methods - Local Surrogate (LIME)

- Using an interpretable model to *mimic* the behavior of an uninterpretable model

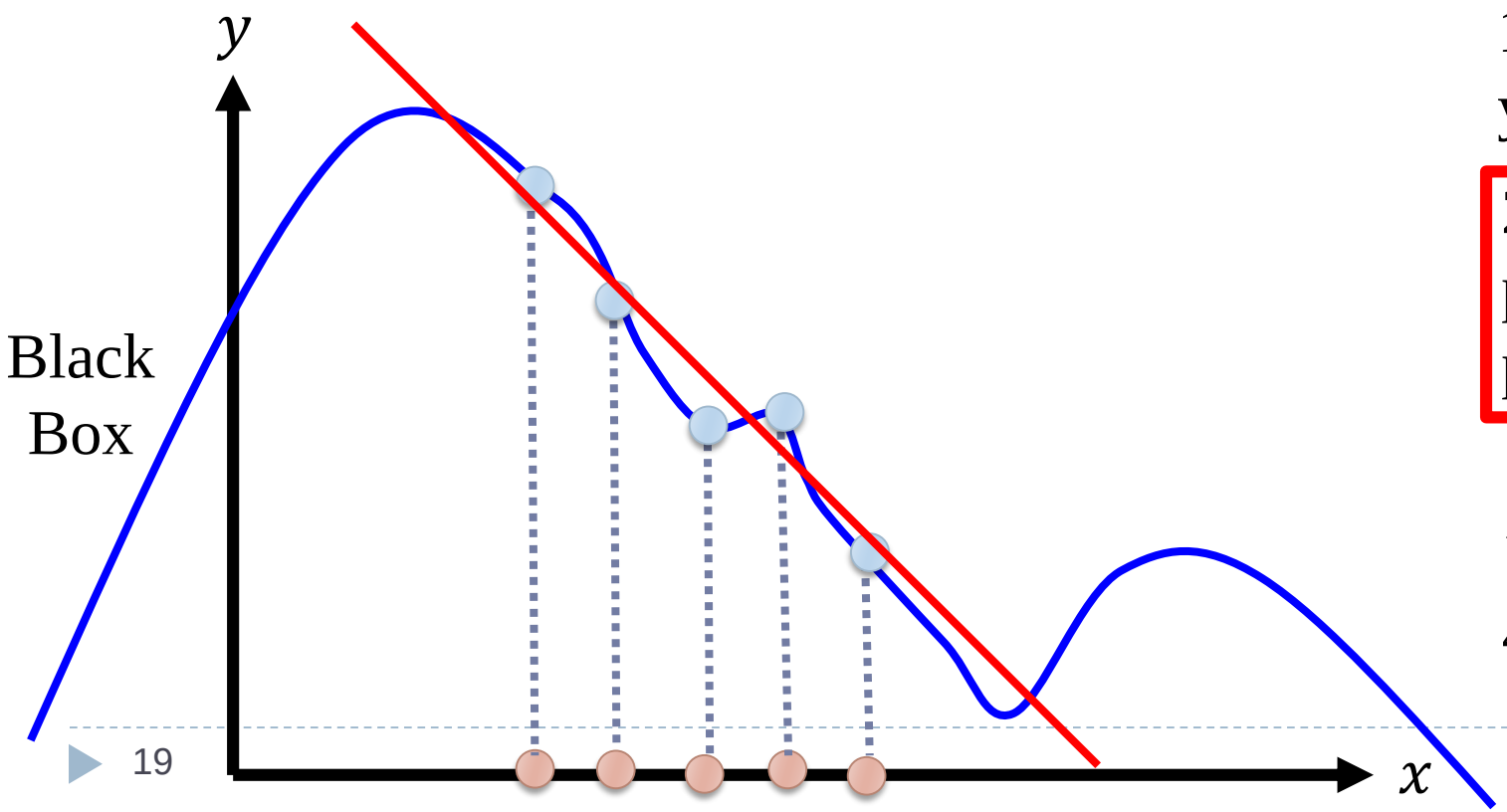



***Problem***: Interpretable model cannot mimic complex one…

However, it can mimic a local region!

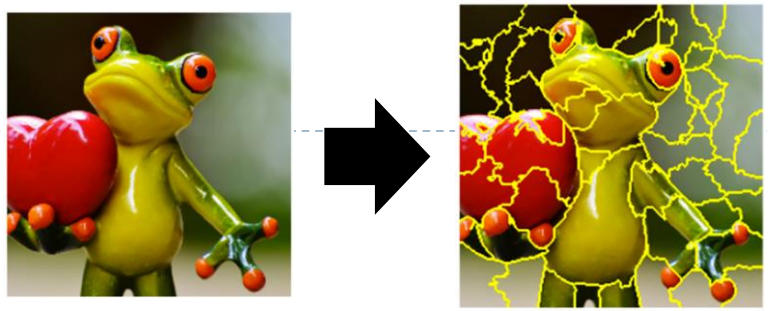
# Local Interpretable Model-Agnostic Explanations (LIME)

- For text and images, the solution is to turn single words or super-pixels on or off. In the case of tabular data, LIME creates new samples by perturbing each feature individually, drawing from a normal distribution with mean and standard deviation taken from the feature

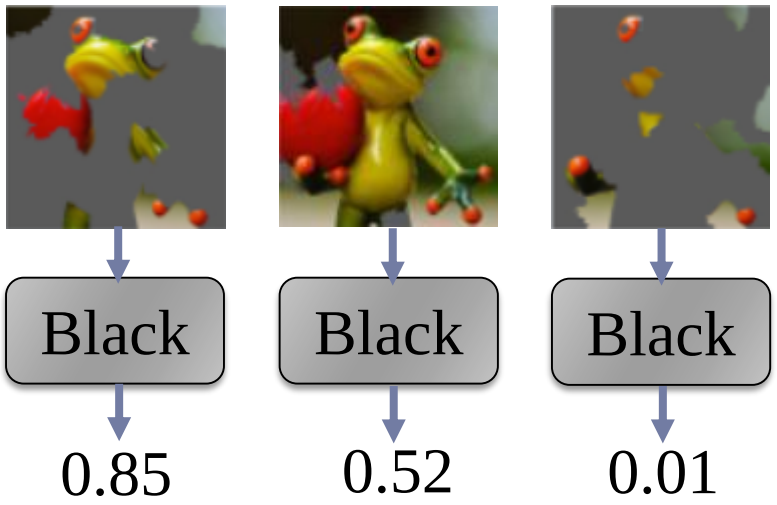



1. Given a data point you want to explain
2. Sample at the nearby points by perturb dataset to get the black box predictions for these new points
3. Fit with interpretable model on the weighted samples
4. Interpret the linear model

# LIME – Image data



1. Given a data point you want to explain
2. Sample at the nearby
   - Each image is represented as a set of superpixels (segments)



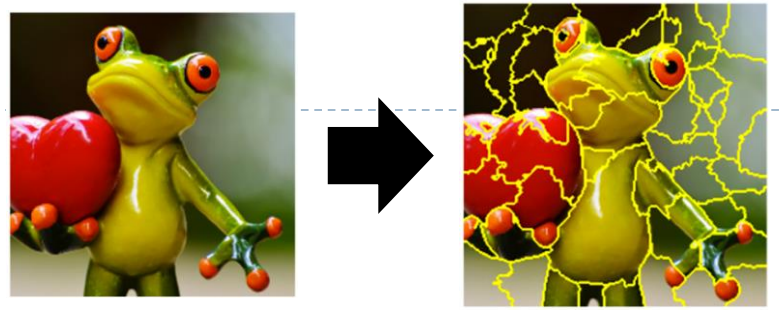

Randomly delete some segments

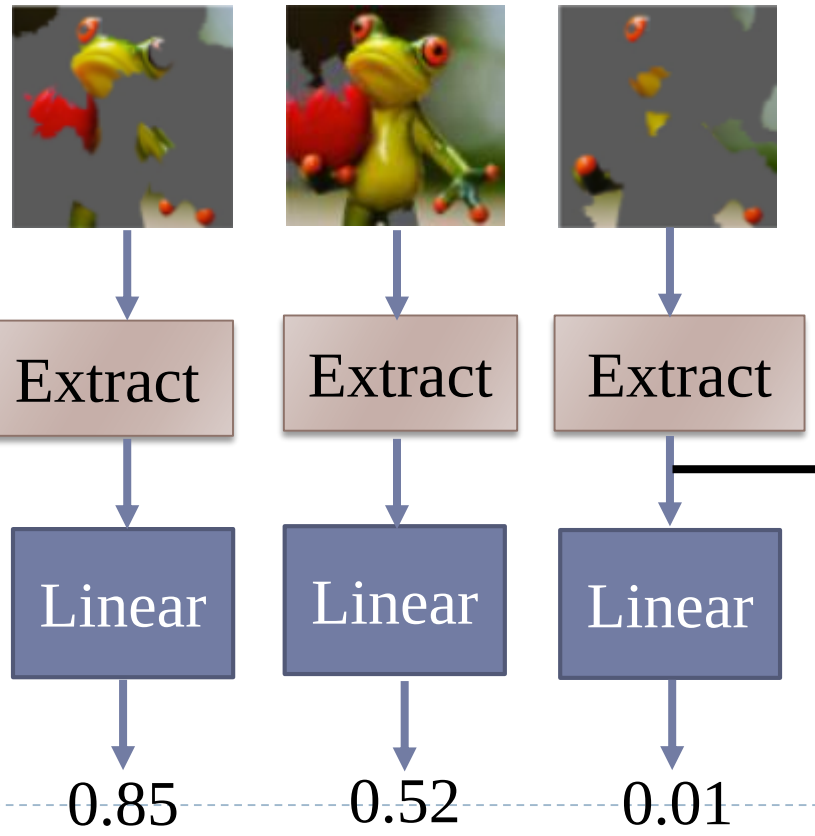
Compute the probability of "frog" by black box

https://medium.com/@kstseng/lime-local-interpretable-model-agnostic-explanation-%E6%8A%80%E8%A1%93%E4%BB%8B%E7%B4%B9-a67b6c34c3f8

# LIME – Image data

3. Fit with linear (or interpretable) model






$x_1 \cdots\cdots x_m \cdots\cdots x_M$

$$x_m = \begin{cases} 0 & \text{Segment } m \text{ is deleted} \\ 1 & \text{Segment } m \text{ exists} \end{cases}$$

$M$ is the number of segments

# LIME – Image data

4. Interpret the model you learned

Extract

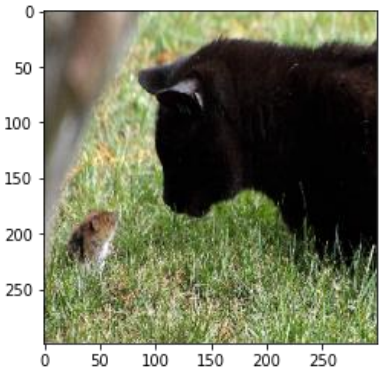
Linear

0.85

$$y = \beta_1 x_1 + \cdots + \beta_m x_m + \cdots + \beta_M x_M$$

$$x_m = \begin{cases} 0 & \text{Segment } m \text{ is deleted} \\ 1 & \text{Segment } m \text{ exists} \end{cases}$$

$M$ is the number of segments



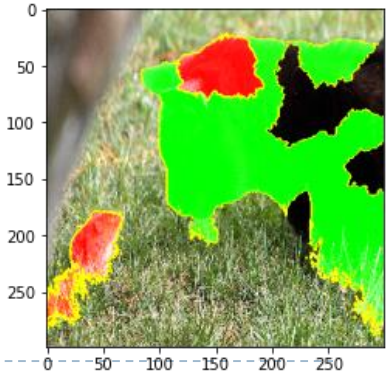

If $\beta_m \approx 0$ ➡ segment $m$ is not related to “frog”

If $\beta_m$ is positive

➡ segment $m$ indicates the image is “frog”

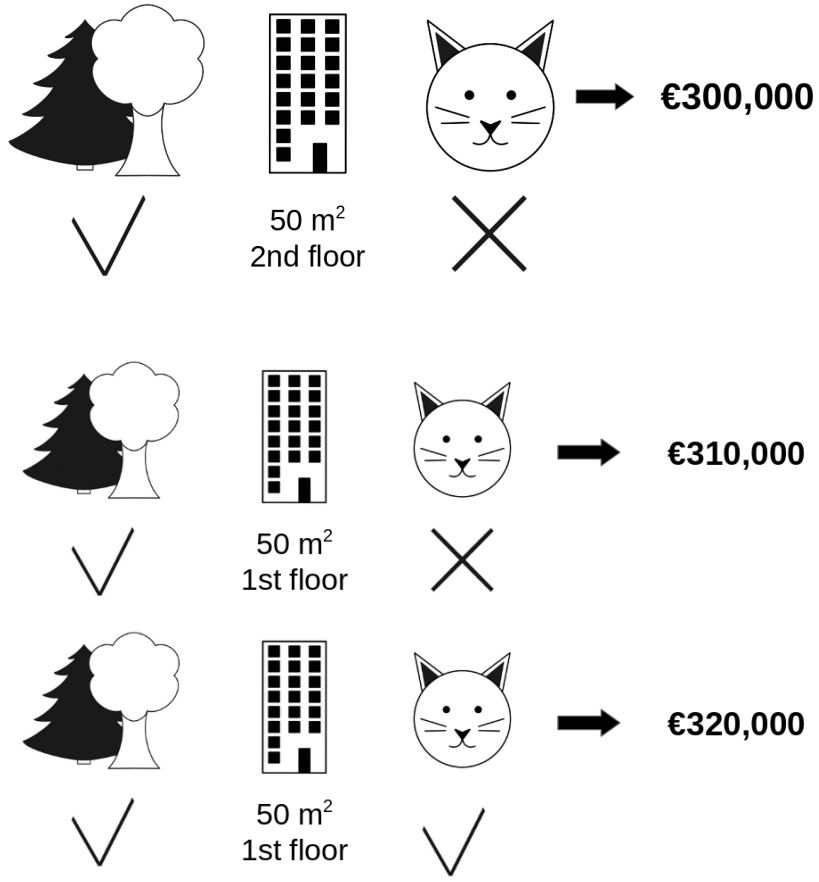
If $\beta_m$ is negative

➡ segment $m$ indicates the image is not “frog”



# Local Methods - SHAP (SHapley Additive exPlanations)

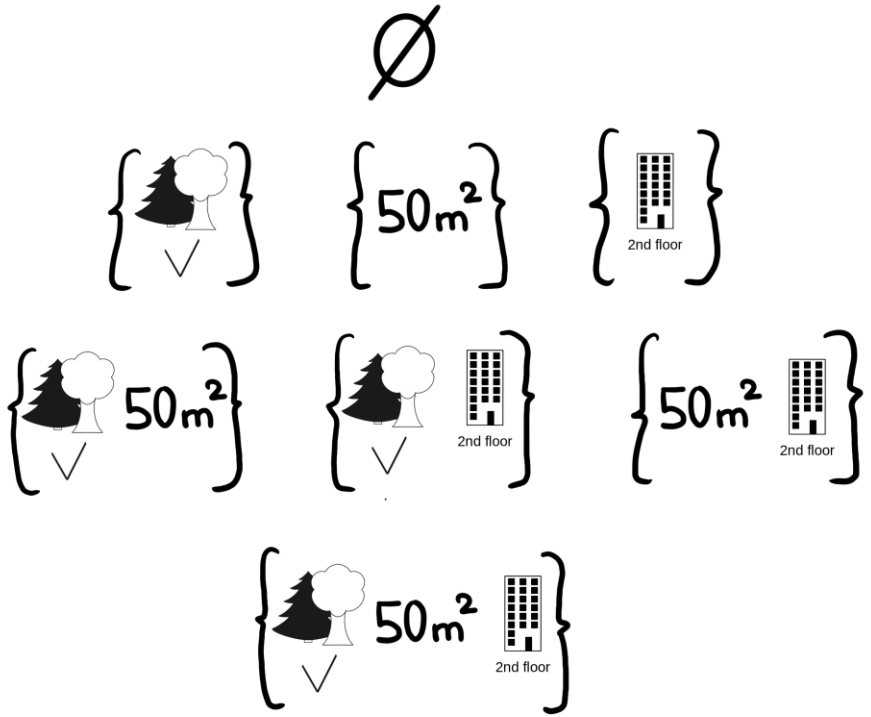
- A prediction can be explained by assuming that each feature value of the instance is a "player" in a game where the prediction is the payout
  - *Shapley values* breakdown a prediction to show the impact of each feature
  - One sample repetition to estimate the contribution of `cat-banned` to the prediction when added to the coalition of `park-nearby` and `area-50`
    - The contribution of `cat-banned` was €310,000 - €320,000 = -€10,000



# Local Methods - SHAP (SHapley Additive exPlanations)

- The figure shows all coalitions of feature values that are needed to determine the Shapley value for cat-banned

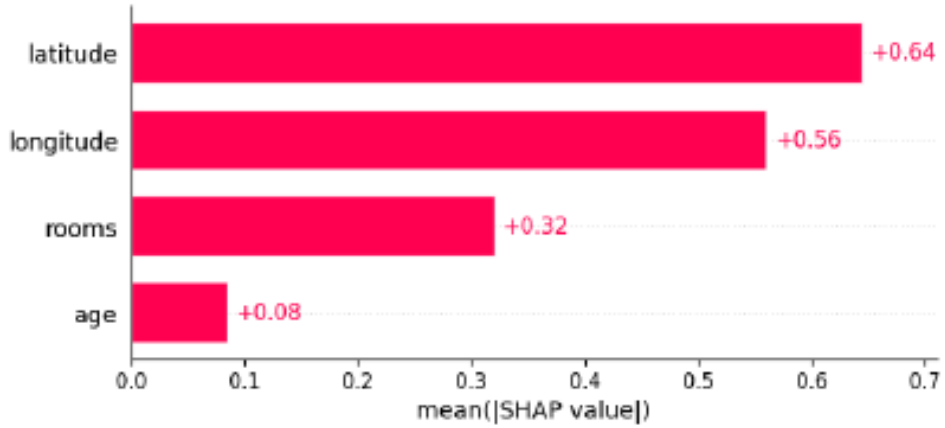
- For each of these coalitions we compute the predicted apartment price with and without the feature value cat-banned and take the difference to get the marginal contribution
  - We replace the feature values of features that are not in a coalition with random values
  - The Shapley value is the (weighted) average of marginal contributions

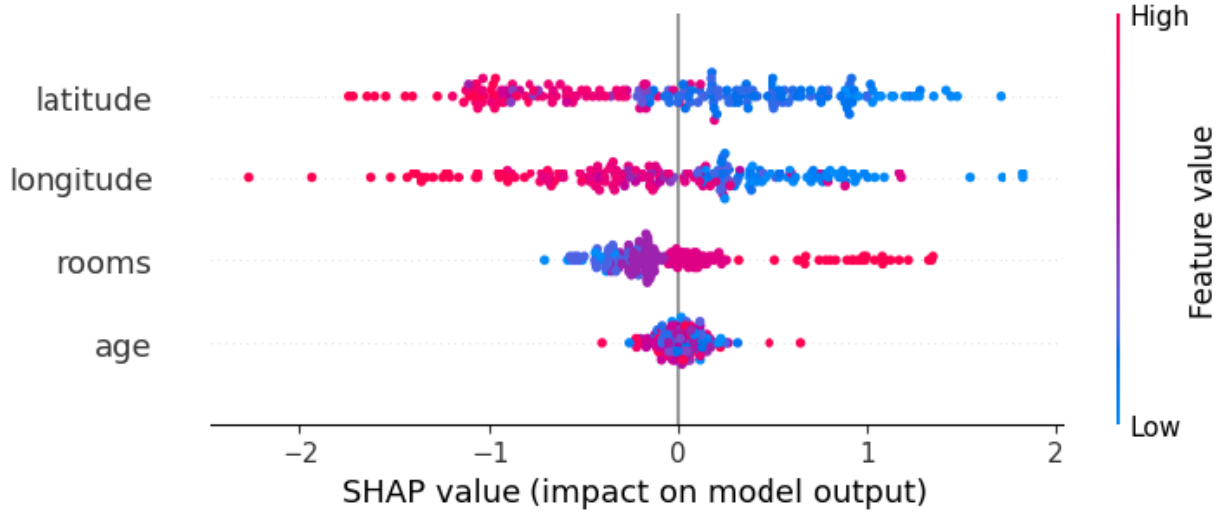


# Local Methods - SHAP (SHapley Additive exPlanations)

- SHAP assign an estimated Shapley value to *individual predictions for each feature*
  - SHAP proposed *KernelSHAP*, an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models. In addition, TreeSHAP, an efficient estimation approach for tree-based models is also proposed
  - The *force plot*, we predicted 2.38, whereas the base value is 2.095
    - The predicted value ($238k) of house is larger than the average $209.5k. Longitude of - 122.3 and, age of 28 increase prediction; latitude of 37.83 and rooms=3 decrease predicted value, but don't cancel out the positive effects

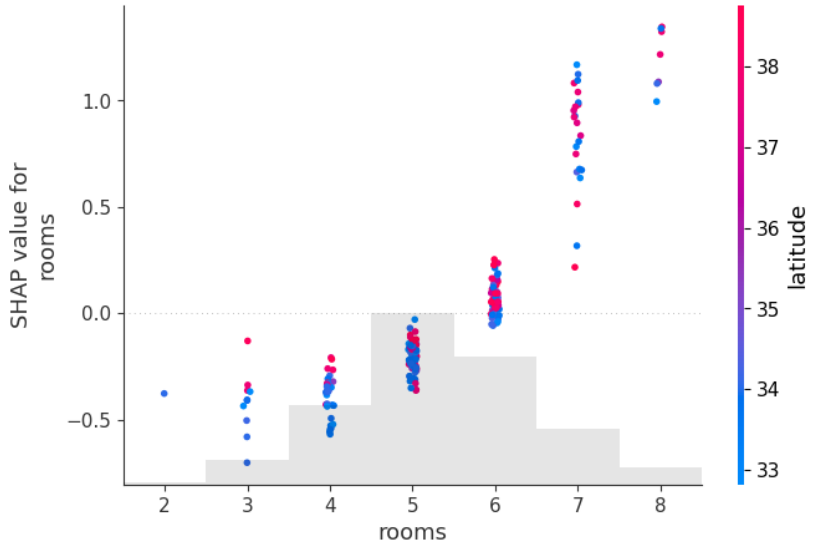
- SHAP has many global methods based on aggregations of Shapley values
  - Permutation importance doesn't tell you how each features matter on each prediction. If a feature has medium permutation importance, that could mean it has a large effect for a few predictions, but no effect in general, or a medium effect for all predictions
- SHAP also provides *importance plots*
- SHAP *summary plots* give us a view of feature importance and what is driving it
  - Each point on the summary plot is a Shapley value for a feature and an instance



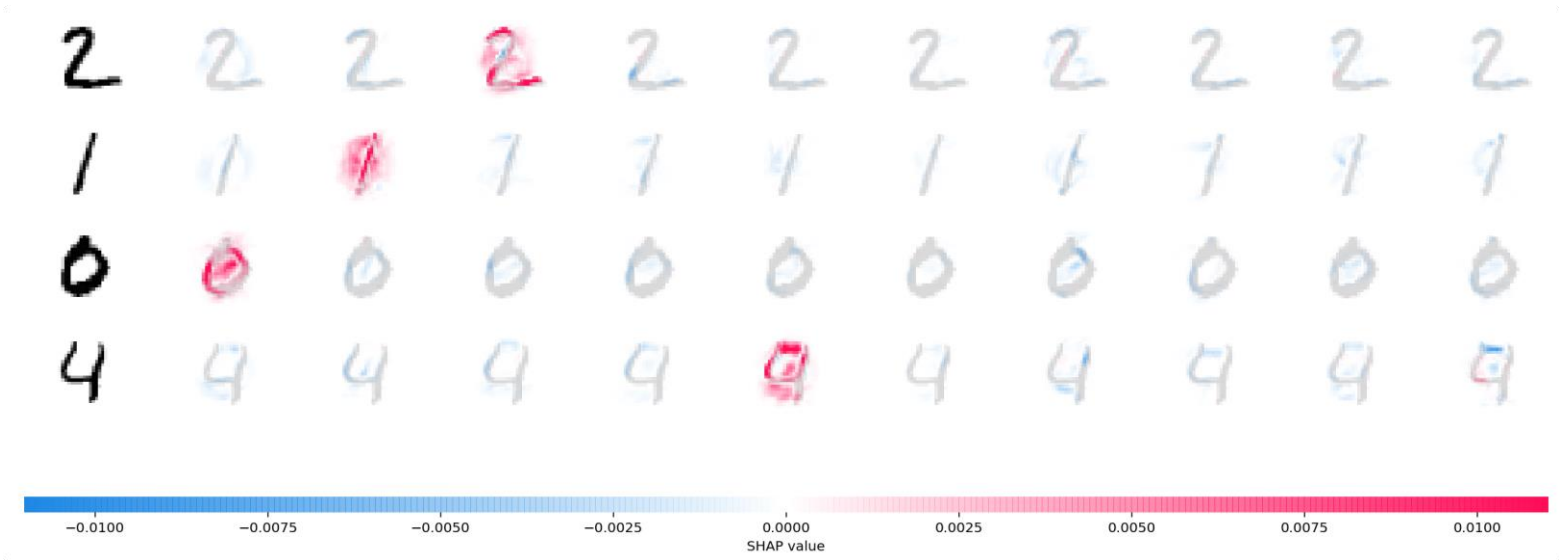
# Local Methods - SHAP (SHapley Additive exPlanations)

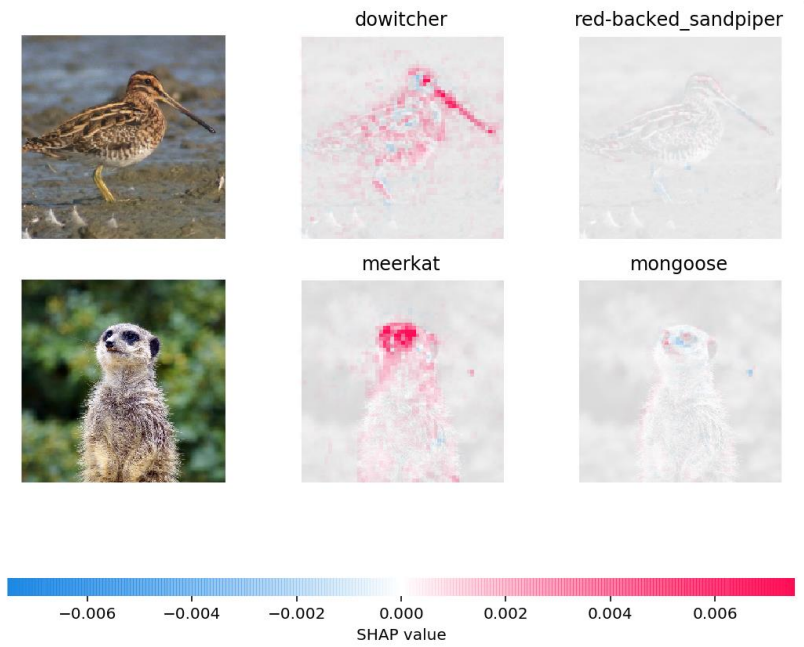
- We've previously used Partial Dependence. But what is the distribution of effects? Is the effect of having a certain value pretty constant, or does it vary a lot depending on the values of other features
  - SHAP *dependence plots/interaction plots* provide a similar insight to PDP's
  - Each dot represents a row of the data. The horizontal location is the actual value from the dataset, and the vertical location shows what having that value did to the prediction



# Local Methods - SHAP (SHapley Additive exPlanations)

- Extended to deep learning models
  - Deep SHAP (DeepLift) is a high-speed approximation algorithm for *Shapley* values
  - GradientShap combines ideas from Integrated Gradients, SHAP, and SmoothGrad into a single expected value equation



# Interpretability in Data-Centric ML

- *Interpretable features* are those that are most useful and meaningful to the user (i.e., whoever will be using the explanations). A few to consider include:
  1. **Readability**: Do users understand what the feature refers to at all? Like X1 are not readable, and should be replaced with natural-language descriptions
  2. **Understandability**: Can users reason about the feature value? For example, thinking about income in direct dollar values ($50,000) is easier than thinking about a normalized measure of income (.67).
  3. **Meaningfulness/Relevancy**: Users have an easier time trusting, and are therefore more likely to use, models that use information that they *believe is important*
  4. **Abstract Concepts**: Depending on user-base, it may be valuable to condense features into digestible abstract concepts — for example, combine detailed information about the area into a single metric of "neighborhood quality". There is a tradeoff here — using abstract concepts can hide important information

# Interpretability in Data-Centric ML

- The table below shows some examples of features and their properties:

| | Area Quality (numeric) | Average House Size (numeric) | Common House Color (categorical) | Normalized Median Income (numeric) | X12 (numeric) |
|---|---|---|---|---|---|
| Readable | ✓ | ✓ | ✓ | ✓ | |
| Understandable | ✓ | ✓ | ✓ | | |
| Relevant | ✓ | ✓ | | | |
| Abstract Concept | ✓ | | | | |

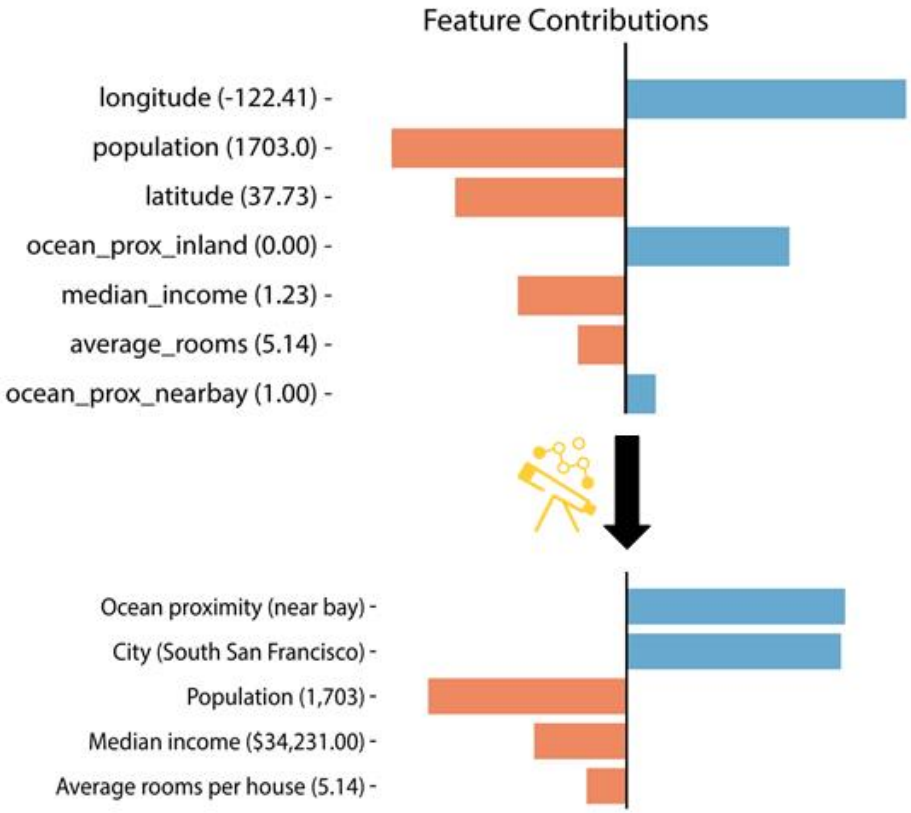
# How do we get interpretable features?

- There are three main approaches to interpretable feature generation:

1. Including users in the feature generation process
   - Traditionally in design tasks, users are included in every step of the process (including feature engineering!) and iterate based on their feedback
   - There are ways to make including the user in feature engineering easier, such as through *collaborative feature engineering* — systems that allow multiple people to easily collaborate in feature generation like Flock or Ballet

# Explanation Transforms

- To generate more interpretable features, we can apply *post-processing* transformations to the explanations themselves
  - Pyreal automatically "undoes" data transformations that reduce the interpretability of features as well as adding additional transformations that can improve the interpretability of features in explanations
  - Notice that one-hot encoded features are combined into a single categorical feature (*ocean proximity*), less interpretable features are transformed to more readable versions (*lat/long -> city*), and features are unstandardized (*median income*)



# Conclusion

- We need an interpretable models!
  - Loan issuers are required by law to explain their models
  - Medical diagnosis model is responsible for human life. Can it be a black box?
  - If a model is used at the court, we must make sure the model behaves in a nondiscriminatory manner
  - If a self-driving car suddenly acts abnormally, we need to explain why
- Make people (your customers, your boss, yourself) comfortable
  - Don’t be afraid about black box model. We do not completely know how brains work! But we trust the decision of humans!
- Several methods you can choose from
  - https://csinva.io/notes/cheat_sheets/interp.pdf

# References

[1] https://christophm.github.io/interpretable-ml-book/

[2] Deep learning with Python, 2nd Edition Chapter 9

[3] https://www.kaggle.com/learn/machine-learning-explainability

[4] https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/xai_v4.pptx

[5] https://github.com/kdd-lab/XAI-Survey

[6] https://christophmolnar.com/books/shap/

# Appendix

# Resources

- Lectures or books
  - https://christophm.github.io/interpretable-ml-book/
  - https://github.com/jphall663/awesome-machine-learning-interpretability
  - https://github.com/wangyongjie-ntu/Awesome-explainable-AI
  - Local Model-Agnostic Methods - Counterfactual Explanations
  - Global Model-Agnostic Methods - Prototypes and Criticisms
  - Speech - https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/xai_v4.pptx
- Tree base visualization
  - https://github.com/parrt/dtreeviz
- Interpretable features
  - https://github.com/sibyl-dev/pyreal

# Resources

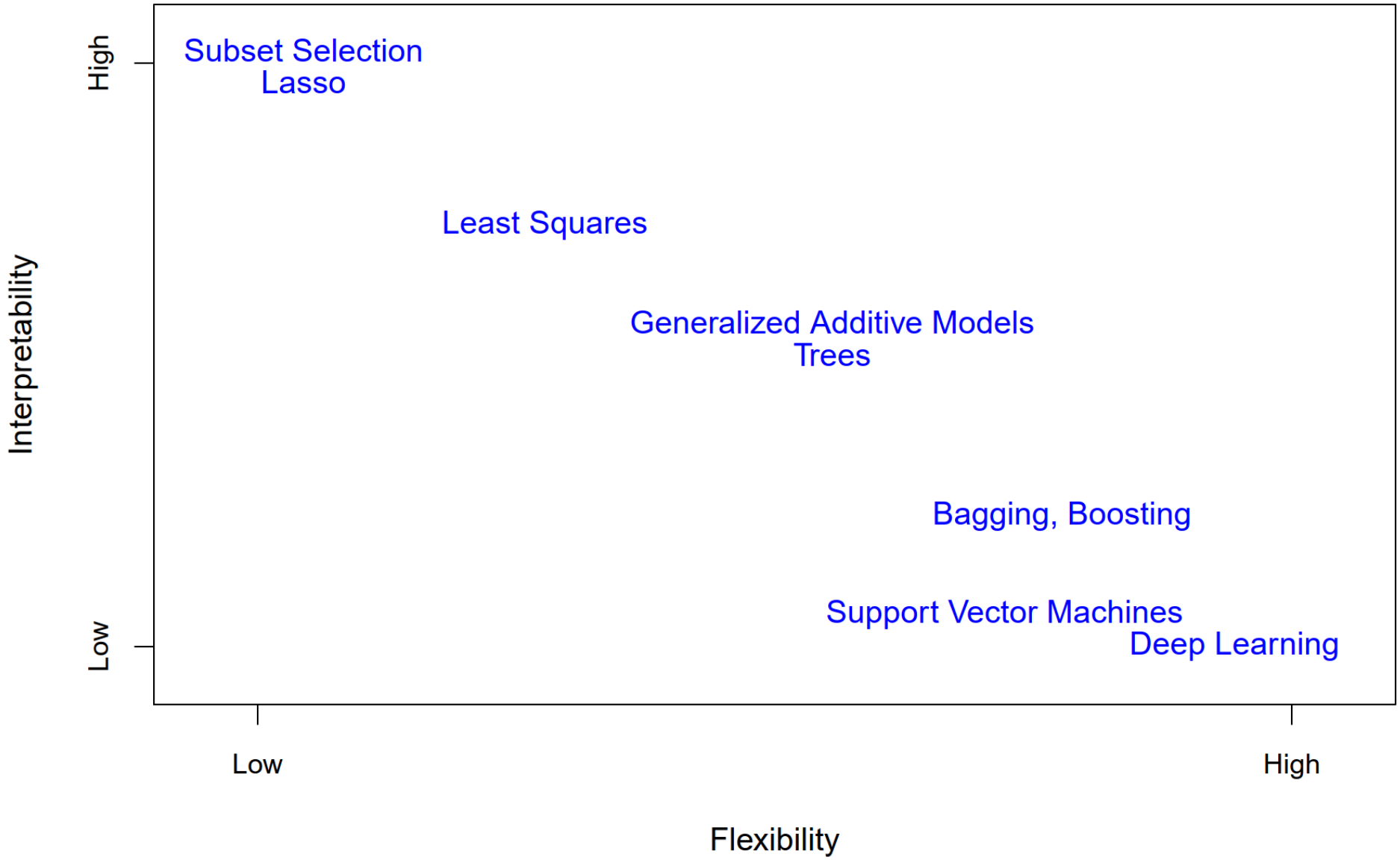
- Decision Rules
  - https://github.com/csinva/imodels
- Local methods
  - https://github.com/slundberg/shap
  - https://github.com/marcotcr/lime
- Unify framework
  - https://github.com/interpretml/interpret
  - https://github.com/SelfExplainML/PiML-Toolbox
- Prototype and counterfactual
  - https://github.com/Trusted-AI/AIX360
  - https://github.com/SeldonIO/alibi
- Casual
  - https://github.com/uber/causalml

# Resources

- Network visualization
  - https://github.com/keisen/tf-keras-vis
  - https://github.com/PAIR-code/saliency
  - https://github.com/sicara/tf-explain
  - https://github.com/pytorch/captum (Also contains LIME and SHAP)

# Some trade-offs when selecting models

- Prediction accuracy versus interpretability
  - Linear models are easy to interpret; thin-plate splines are not.
  - Parsimony versus black-box
    - We often prefer a simpler model involving fewer variables over a black-box predictor involving them all.
  - Good fit versus over-fit or under-fit
    - How do we know when the fit is just right?


Subset Selection
Lasso
Least Squares
Generalized Additive Models
Trees
Bagging, Boosting
Support Vector Machines
Deep Learning
High
Low
Interpretability
Low
High
Flexibility

# Example-Based Explanations

- Example-based explanation methods select particular instances to explain the behavior of models or to explain the underlying data distribution
  - The difference to model-agnostic methods is that the example-based methods explain a model by selecting instances of the dataset and not by creating summaries of features (such as feature importance or partial dependence)
  - Example-based methods work well if the feature values of an instance carry more context, meaning the data has a structure, like images or texts do
- For instance
  - A physician sees a patient with an unusual cough and a mild fever. The symptoms remind her of another patient she had years ago with similar ones. She suspects that the patient could have the same disease and she takes a blood sample to test for this specific disease
  - A data scientist works on a new project for one of his clients: Analysis of the risk factors that lead to the failure of production machines for keyboards. The data scientist remembers a similar project he worked on and reuses parts of the code from the old project

# Example-Based Explanations

- These stories illustrate how we humans think in examples or analogies. The blueprint of example-based explanations is
  - Thing B is similar to thing A and A caused Y, so I predict that B will cause Y as well
  - Implicitly, decision trees and KNN works like a example-based method
- The following interpretation methods are all example-based:
  - **Prototypes** are a selection of representative instances from the data and criticisms are instances that are not well represented by those prototypes
  - **Counterfactual explanations** tell us how an instance has to change to significantly change its prediction. By creating counterfactual instances, we learn about how the model makes its predictions and can explain individual predictions
  - **Influential instances** are the training data points that were the most influential for the parameters of a prediction model or the predictions themselves. Analyzing influential instances helps to find problems with the data and debug the model

# Measurement bias

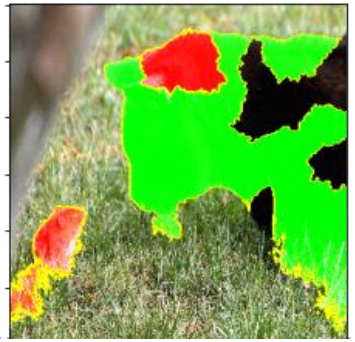
- Measurement bias can also be quite subtle. Perhaps all the photographs of foxes are taken against snow, whereas all the photographs of dogs are against grass. A machine learning model may learn to discriminate between snow and grass and achieve superior accuracy to a model that actually learns the features of foxes and dogs. So, we need to be mindful of what other things are within our images

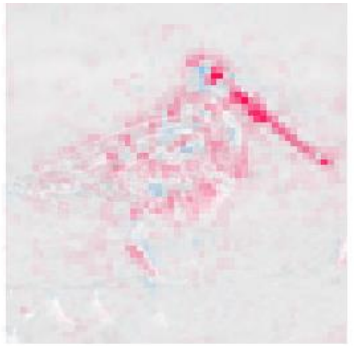
# Neural Network Interpretation

- It's often said that deep learning models are "black boxes": they learn representations that are difficult to extract and present in a human-readable form. Although this is partially true for certain types of deep learning models, it's definitely not true for convnets
  - The representations learned by convnets are highly amenable to visualization, in large part because they're representations of visual concepts. Since 2013, a wide array of techniques has been developed for visualizing and interpreting these representations
    - Neural networks learn features and concepts in their hidden layers and we need special tools to uncover them
    - The gradient can be utilized to implement interpretation methods that are more computationally efficient than model-agnostic methods that look at the model "from the outside"

# Neural Network Interpretation

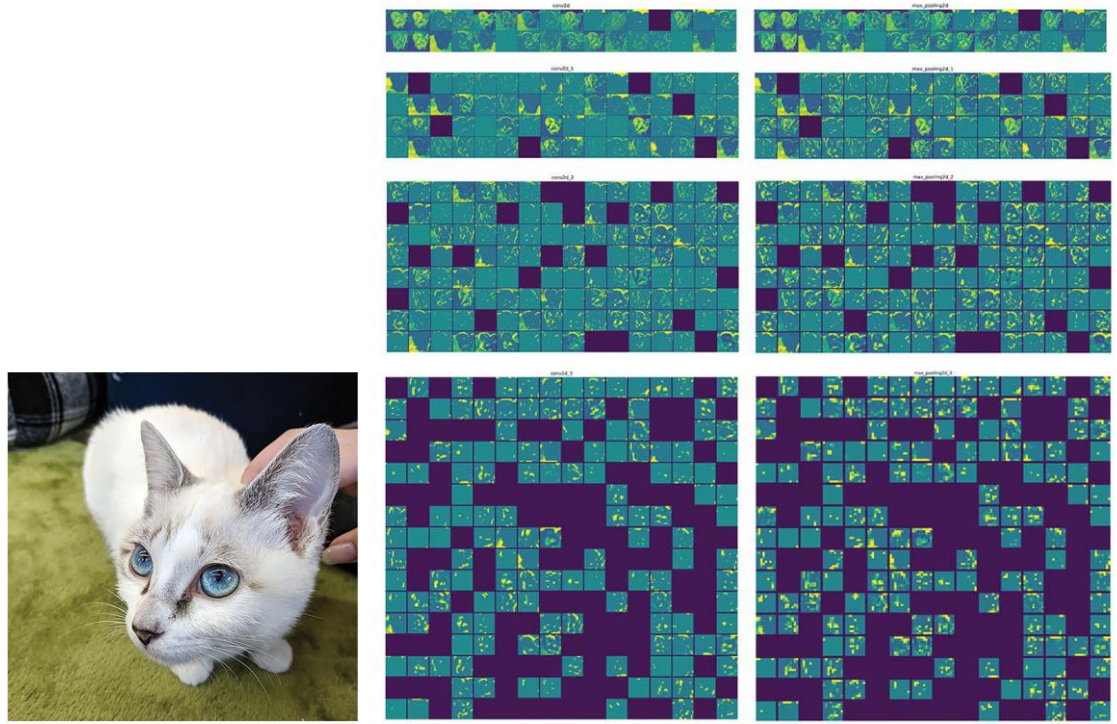
- **Visualizing intermediate convnet outputs** (intermediate activations) **-** Useful for understanding how successive convnet layers transform their input, and for getting a first idea of the meaning of individual convnet filters
- **Visualizing convnet filters -** Useful for understanding precisely what visual pattern or concept each filter in a convnet is receptive to
- **Visualizing heatmaps of class activation in an image -** Useful for understanding which parts of an image were identified as belonging to a given class, thus allowing you to localize objects in images





# Visualizing intermediate activations

- There are a few things to note here:
  - The first layer acts as a collection of various *edge detectors*. At that stage, the activations retain almost all of the information present in the initial picture
  - As you go deeper, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts such as “cat ear” and “cat eye.”
  - The sparsity of the activations increases with the depth of the layer: in the first layer, almost all filters are activated by the input image, but in the following layers, more and more filters are blank. This means the *pattern encoded by the filter isn’t found* in the input image

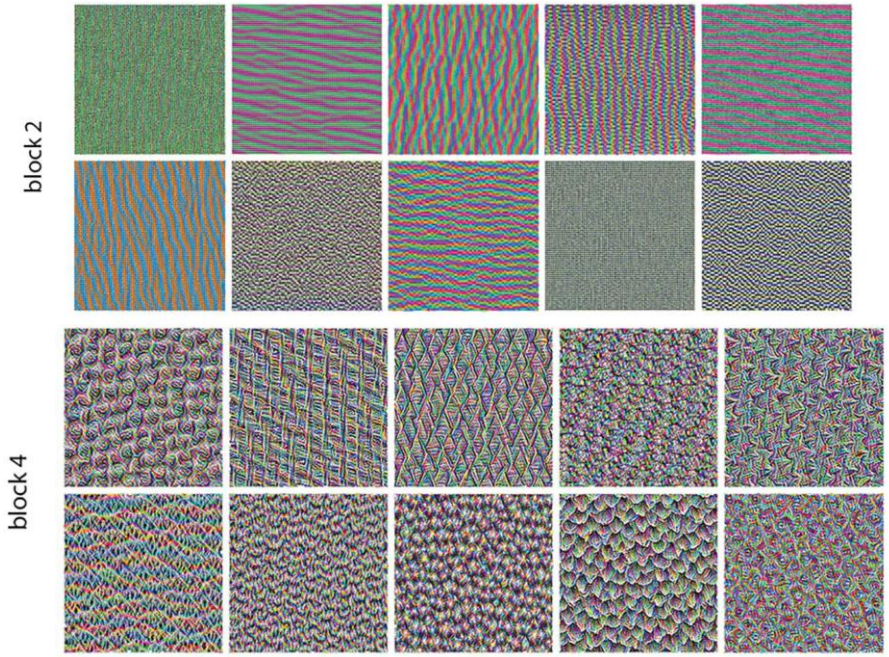


# Visualizing intermediate activations

- Visualizing intermediate activations consists of displaying the values returned by various convolution and pooling layers in a model, given a certain input
  - We have just evidenced an important universal characteristic of the representations learned by DNNs
    - The features extracted by a layer become increasingly abstract with the depth of the layer
    - The activations of higher layers carry less and less information about the specific input being seen, and more and more information about the target
  - A DNN effectively acts as an information distillation pipeline, with raw data going in (in this case, RGB pictures) and being repeatedly transformed so that irrelevant information is filtered out (for example, the specific visual appearance of the image), and useful information is magnified and refined (for example, the class of the image)

# Visualizing convnet filters

- Another easy way to inspect the filters learned by convnets is to display the visual pattern that each filter is meant to respond to
  - This can be done with *gradient ascent in input space*: applying gradient ascent to the value of the input image of a convnet so as to maximize the response of a specific filter, starting from a blank input image. The resulting input image will be one that the chosen filter is maximally responsive to

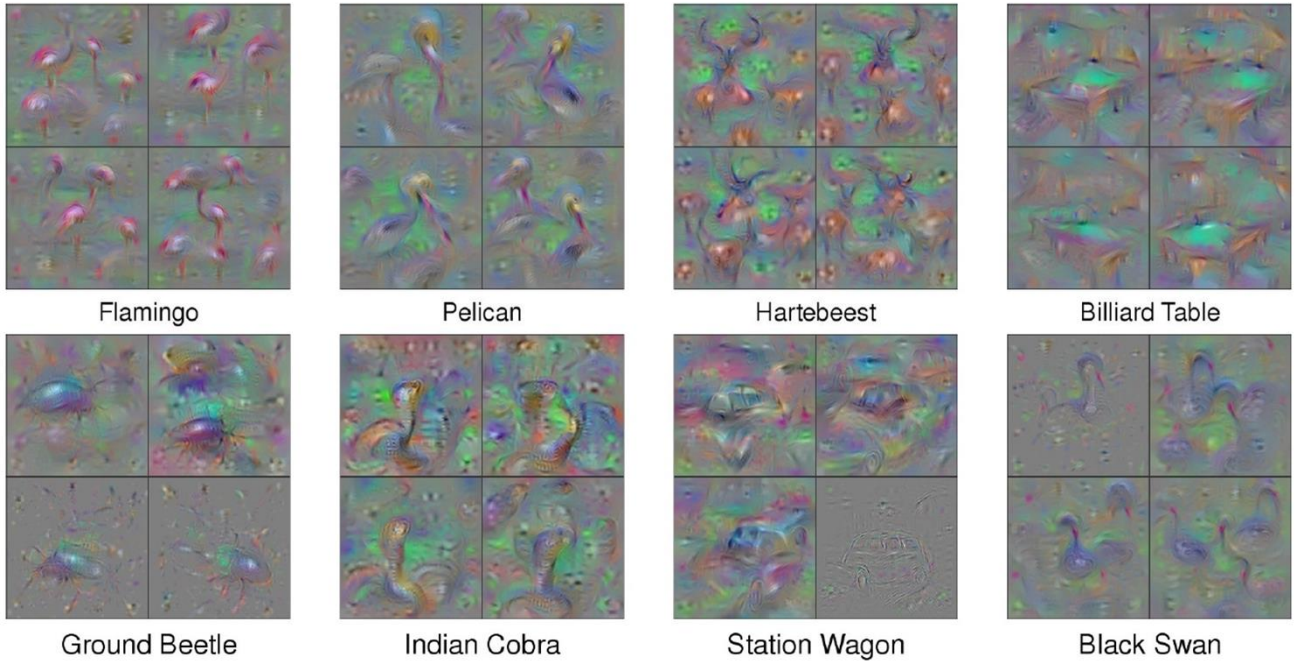



$$img^* = arg\max_{img} \sum_{x,y} h_{n,x,y,z}(img)$$

# Visualizing convnet filters

- These filter visualizations tell you a lot about how convnet layers see the world
  - The filters from the first layers in the model encode simple directional edges and colors (or colored edges, in some cases). The filters from layers a bit further up the stack, such as in block4, encode simple textures made from combinations of edges and colors
  - The filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on
  - You could also perform $img^* = arg\max_{img} y_i$ to ask the network what a class looks like
    - This may require some constraints

https://arxiv.org/abs/1506.06579



# Visualizing heatmaps of class activation

- We'll introduce one last visualization technique—one that is useful for understanding which parts of a given image led a convnet to its final decision
  - This general category of techniques is called class activation map (CAM) or saliency maps visualization, and it consists of producing heatmaps of class activation over input images.
  - Occlusion- or perturbation-based: Methods like SHAP and LIME manipulate parts of the image to generate explanations (model-agnostic)
  - Gradient-based: Many methods compute the gradient of the prediction (or classification score) with respect to the input features. The gradient-based methods (of which there are many) mostly differ in how the gradient is computed

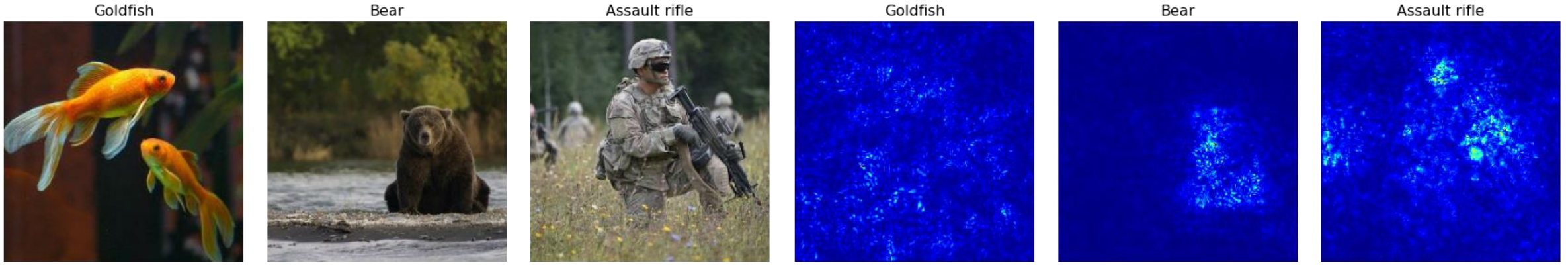
# Visualizing heatmaps of class activation

- The idea of Vanilla Gradient, introduced by Simonyan et al. in 2013
  1. Perform a forward pass of the image of interest
  2. Compute the gradient of class score of interest with respect to the input pixels:

$$E_{grad}(I_0) = \frac{\partial y_c}{\partial I}|_{I=I_0}$$ which means we approximate $$y_c(I) \approx E_{grad}(I)^T I + b$$

  3. Visualize the gradients. You can either show the absolute values or highlight negative and positive contributions separately

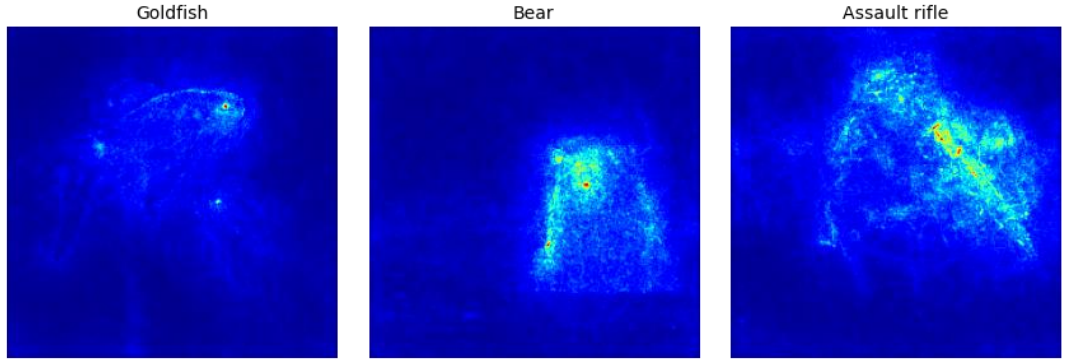
- Vanilla Gradient has a saturation problem



# Visualizing heatmaps of class activation

- Other alternatives
  - SmoothGrad, Integrated Gradients, Grad-CAM, Grad-CAM++, ScoreCAM, etc.

SmoothGrad




Grad-CAM++


Grad-CAM


ScoreCAM

# Interpretable Feature Generation

- Some automated feature engineering algorithms are especially formulated to generate more interpretable features
  - These algorithms often consider things like what information is most *contrastive*, meaning it specifically separates classes, or focus on reducing the feature number to a more easily parsable subset
  - One example of such an algorithm is the Mind the Gap (MTG) algorithm that aims to reduce a set of boolean features to a smaller, interpretable subset
  - See here for more details