Machine Learning from Idea to Production

Keynote, Big Data Conference Vilnius, November 2018

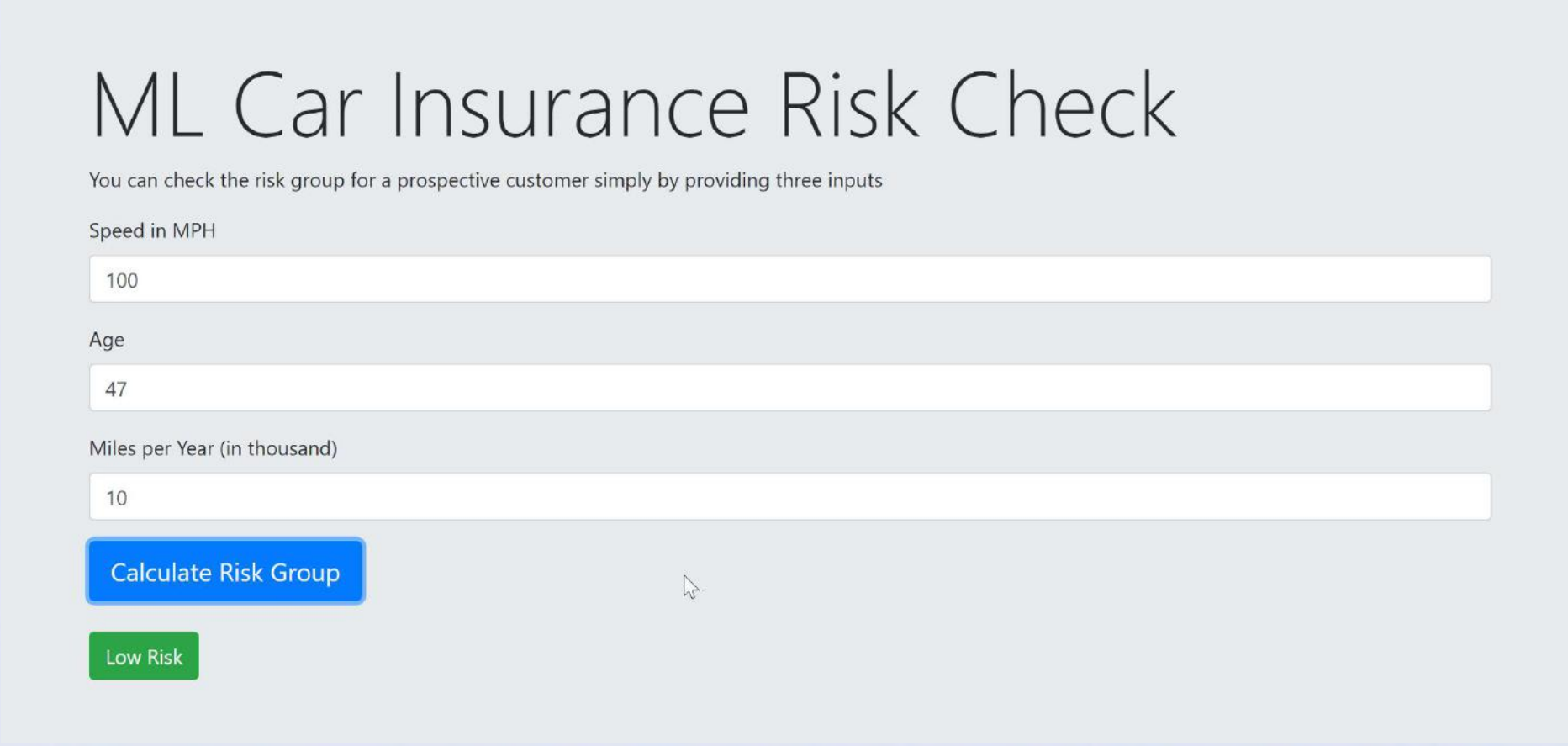
Oliver Zeigermann / @DJCordhose

http://bit.ly/steps-keynote

Objective

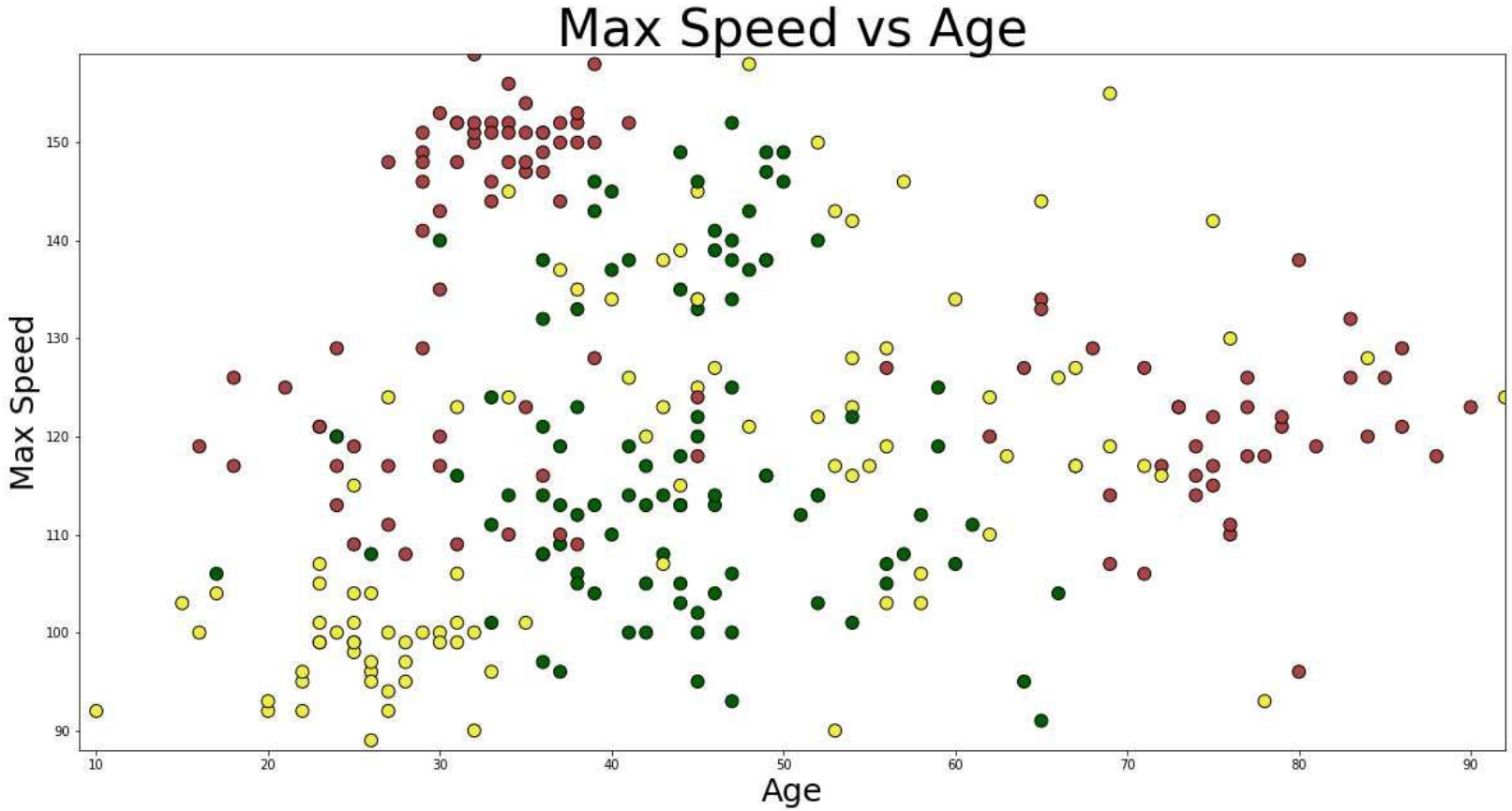
Imagine we are in the car insurance business and want a system that predicts the risk of accidents for prospective customers

ML Car Insurance Risk Calculator



https://djcordhose.github.io/ai/html/calculator.html

Customer Data - Risk of Accidents



How would you rank me (47) for a car having 100 mph top speed, driving 10k miles per year?

Our Process

1. Data Preparation
2. Training
3. Evaluation
4. Production
5. Post-Production

Part I

Data Preparation

- Collection
- Cleaning
- Analysis

Data is King

collecting data might be the hardest part of the job

- but also the most important
- no data, no supervised machine learning
- if you have a simulator, reinforcement learning might be an option

Basic Check on Collected Data

```
[ ]  df.describe()
```

	speed	age	miles	group
count	1500.000000	1500.000000	1500.000000	1500.000000
mean	122.492667	44.980667	30.434000	0.998667
std	17.604333	17.130400	15.250815	0.816768
min	68.000000	16.000000	1.000000	0.000000
25%	108.000000	32.000000	18.000000	0.000000
50%	120.000000	42.000000	29.000000	1.000000
75%	137.000000	55.000000	42.000000	2.000000
max	166.000000	100.000000	84.000000	2.000000

Clean Data and Select Features

speed	age	miles	state	group
98	44	25		1
118	54	24		1
111	26	34	Colorado	0
111	26	34	Colorado	0
114	38	22	California	1
130	55	34		0
118	40	51		0
143	42	34	New York	1
120	41	42		2
148	33	53	California	0
118	48	25		1
105	51	21	Oklahoma	1

speed	age	miles	state	group
115	51	4		1
110	30	28	Texas	2
107	41		Texas	1
123	77	26		0
115	34	22		1
144	48	35		1
93	20	10	California	2
110	32	34		0
127	28	25	Florida	0
1040	48	28		1
112	37	28	Califorina	0
151	34	50		0

Results: Data Cleaning und Feature Selection

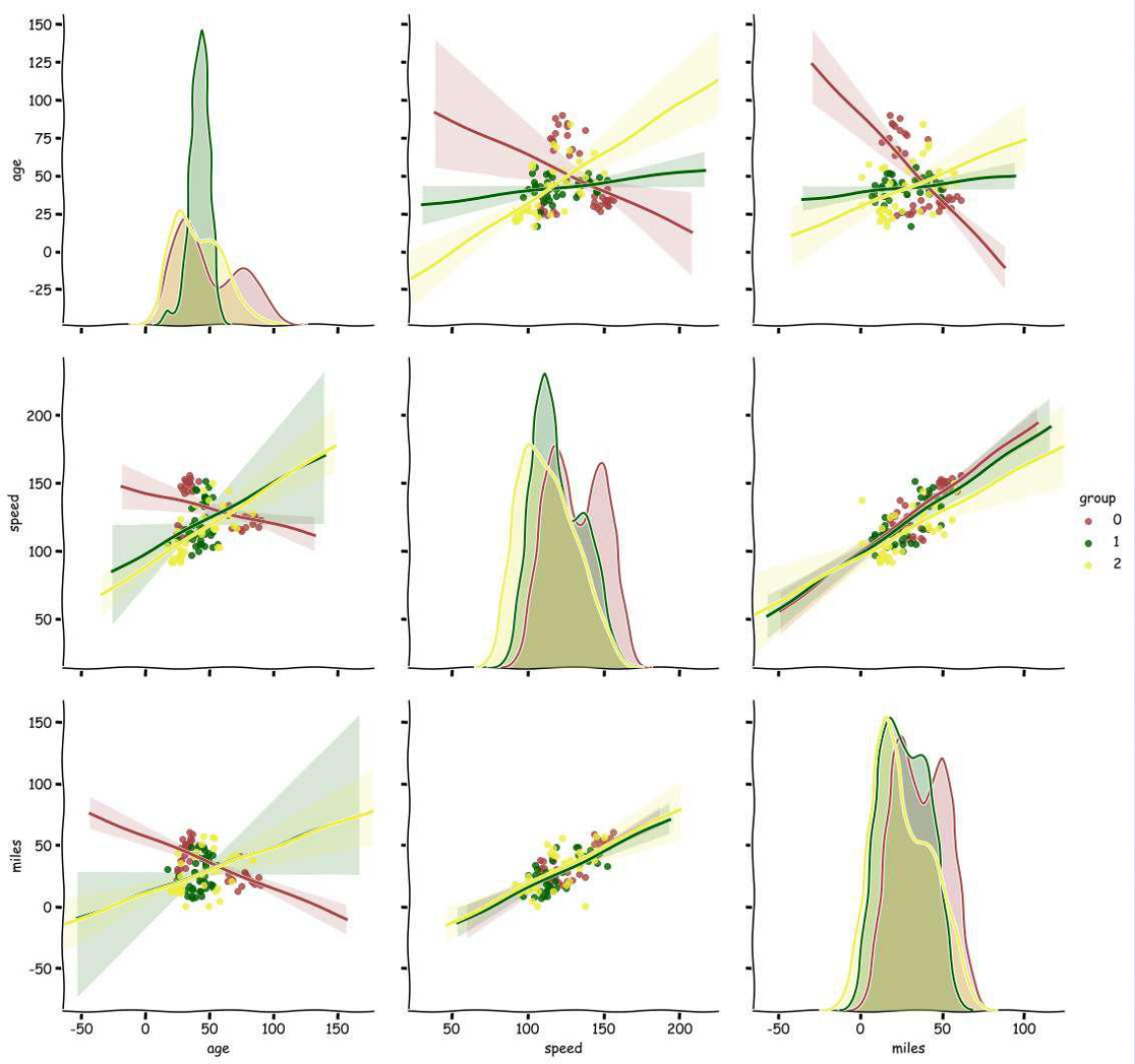
Data Cleaning

- Typos: Califorina
- Outliers: Delete line or replace with decent value
- Doubles: Delete
- Missing Value: Delete line or replace with imputed value

Feature Selection

- Make sure which value to predict
- Row missig more than 50% of values: do not use
- Explore dependencies to decide what use for training input

Exploration



Part II

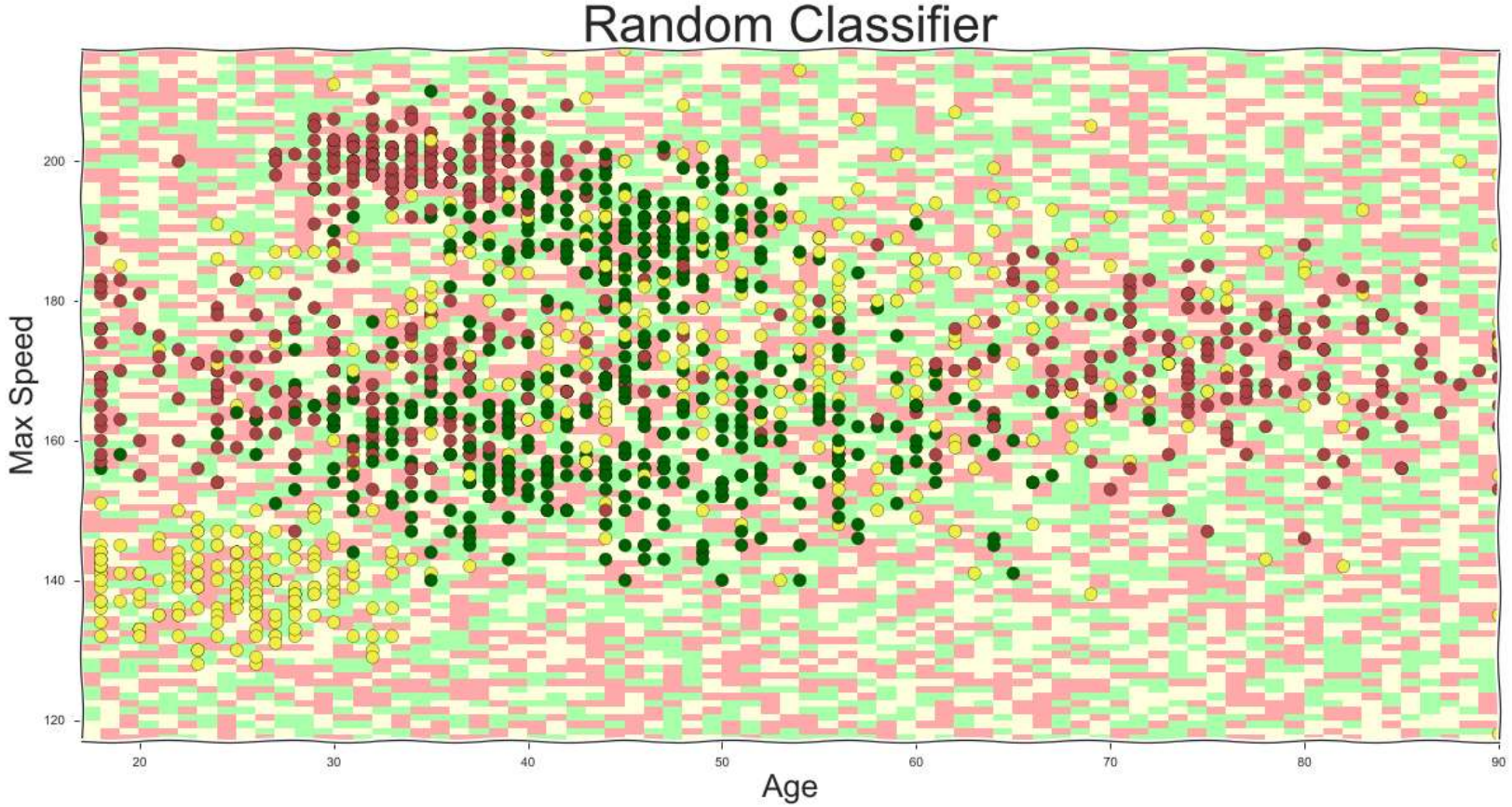
Training

Baselines

- we must be careful not to fool ourselves
- we can never be quite sure what is the highest possible score
- but we can provide a baseline of a prediction with very little effort
- every model must be matched to this baseline

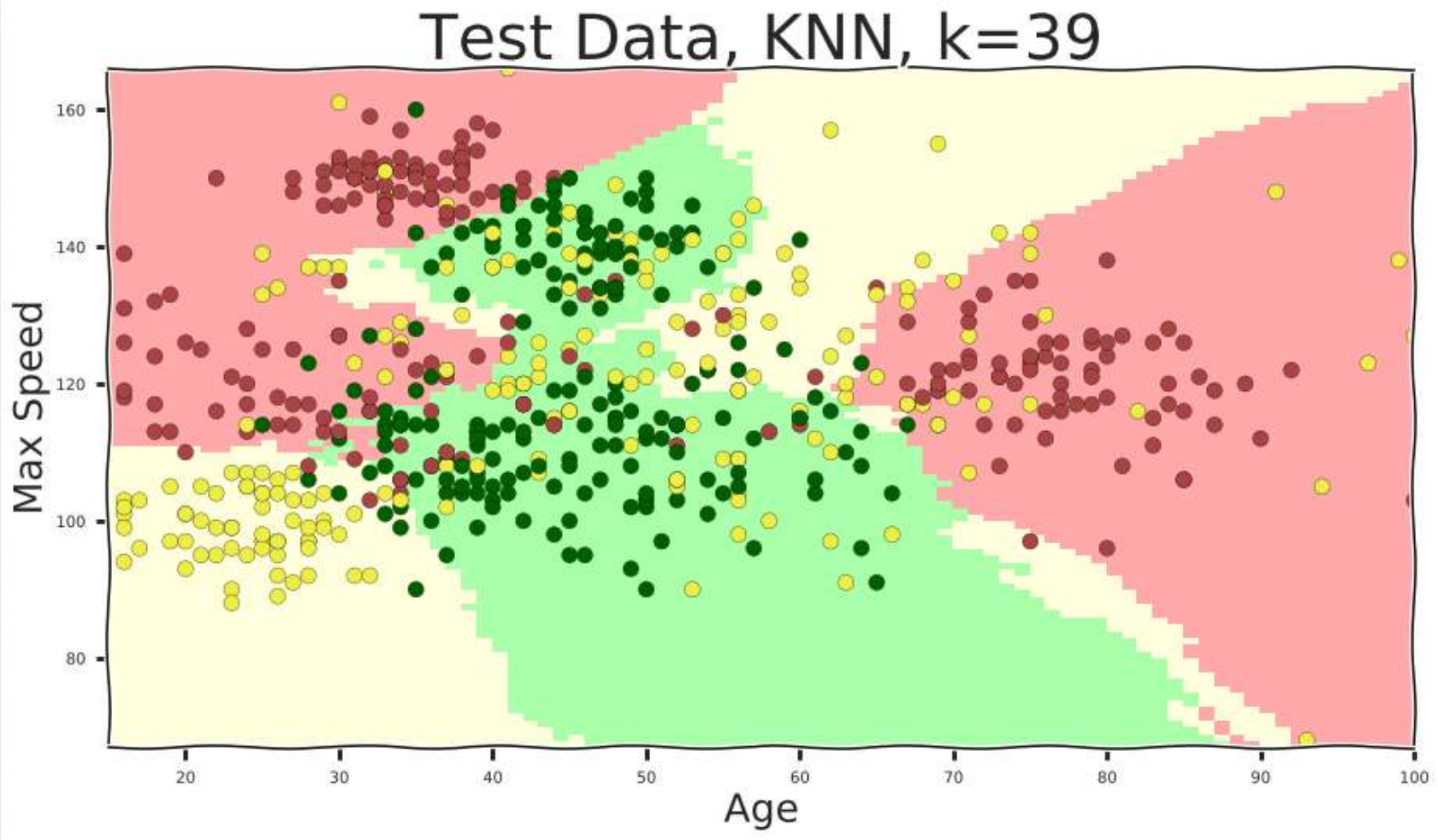
https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa
https://blog.insightdatascience.com/how-to-deliver-on-machine-learning-projects-c8d82ce642b0

Baseline 1: Random



It still gets 33% percents right

Baseline 2: KNN, n=39

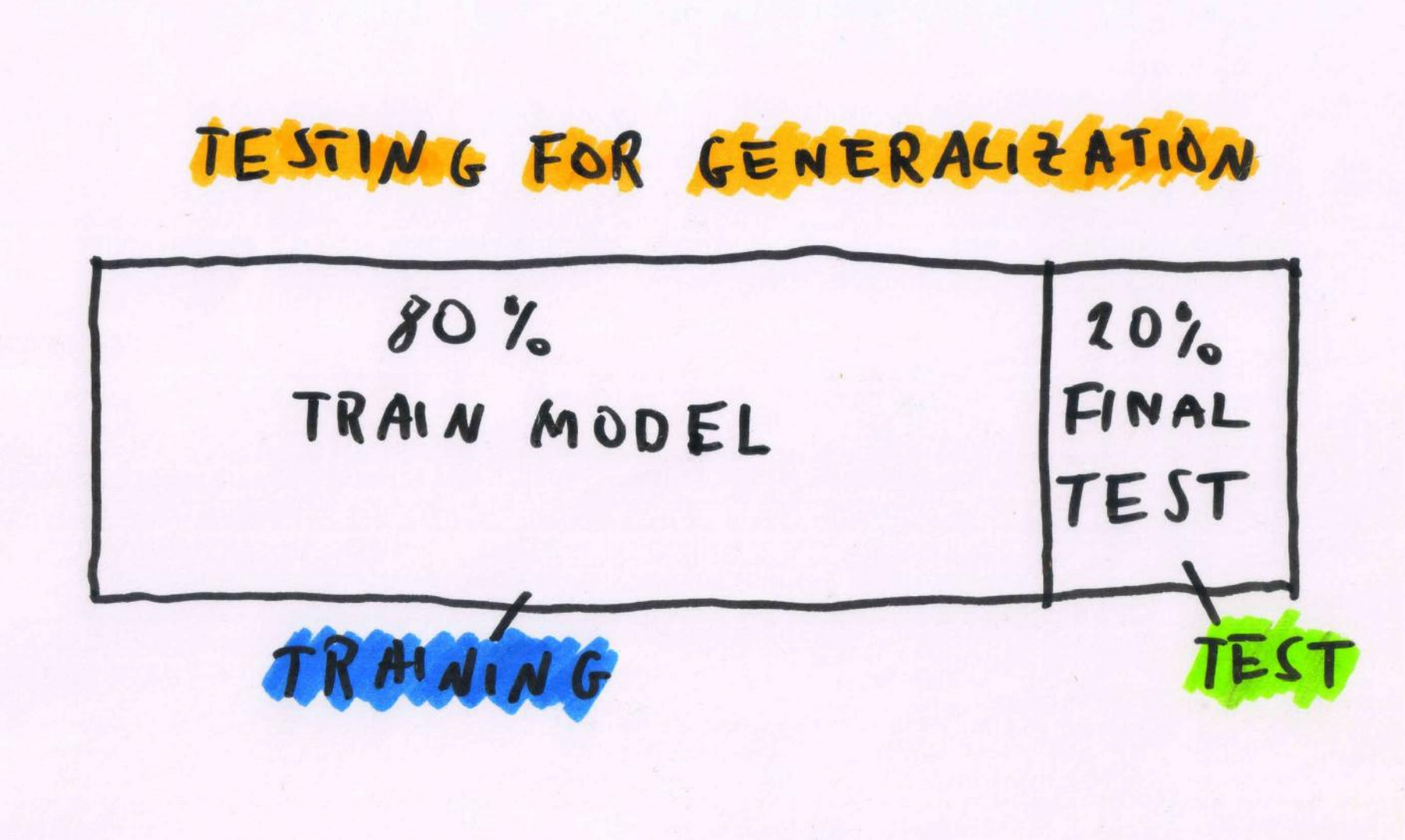


Already 70% accuracy on test data, found using grid search with cross validation

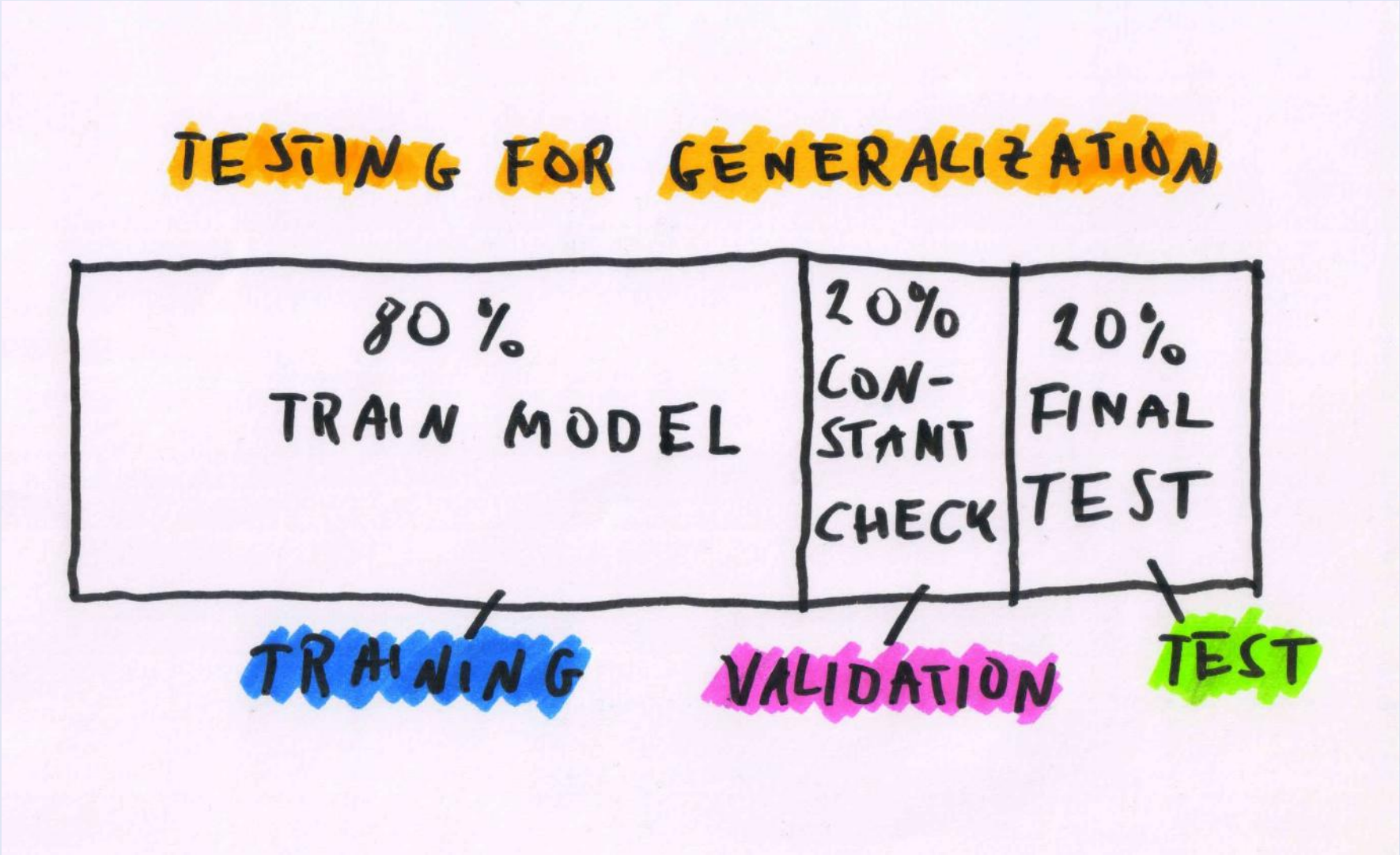
Generalization

- The most important property of a model is if it generalizes well to unknown data
- A machine learning model is of no use if it only works well on the data it has been trained on
  - If it was, the easiest way to achieve this would be a dictionary translating from a set of inputs to the known output
- Conceptually it is a little bit hard to optimize for something you do not know
- So, we introduce a little trick here

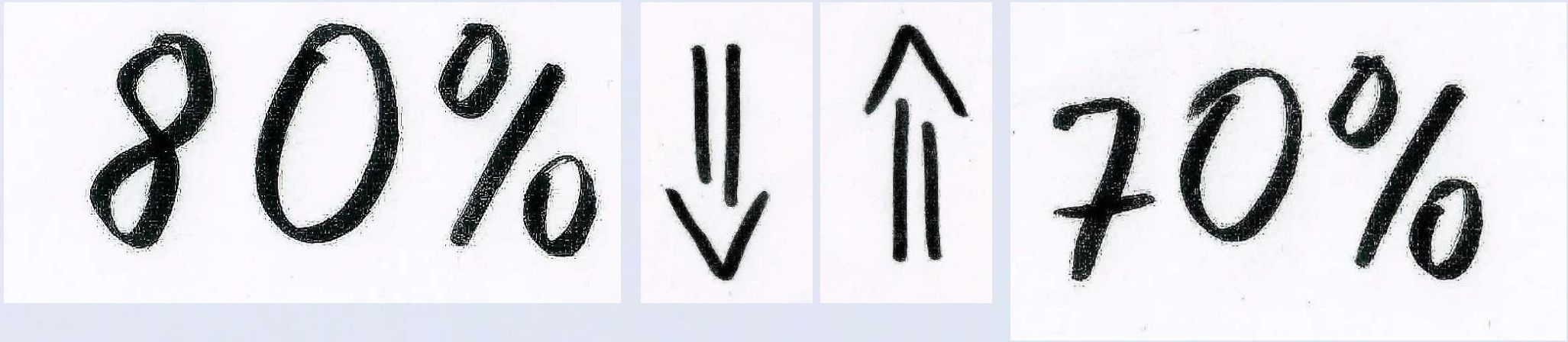
Split known data into training and test



Use some training data for validation



The Issue: Overfitting



Training Score

Test Score

Training and test scores clearly divert

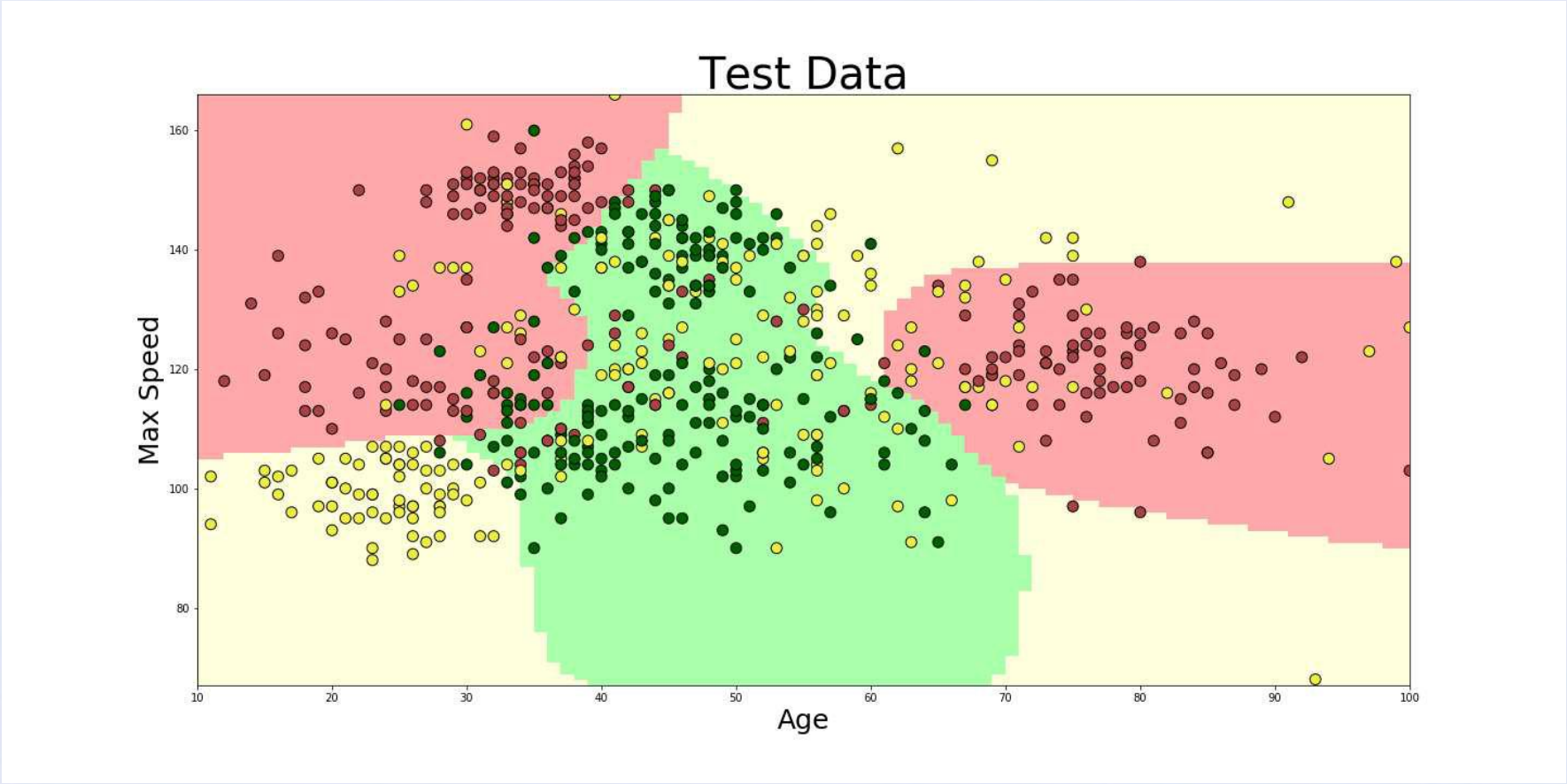
Regularization

Process to counter overfitting

Each ML strategy has its own means of Regularization, e.g.

- KNN: more neighbors
- Decision Trees: reduce depth, use ensenbles
- SVM: gamma and cost
- NN: Dropout, Batch Normalization, Reduced Capacity, Reduced Training Time

Great results

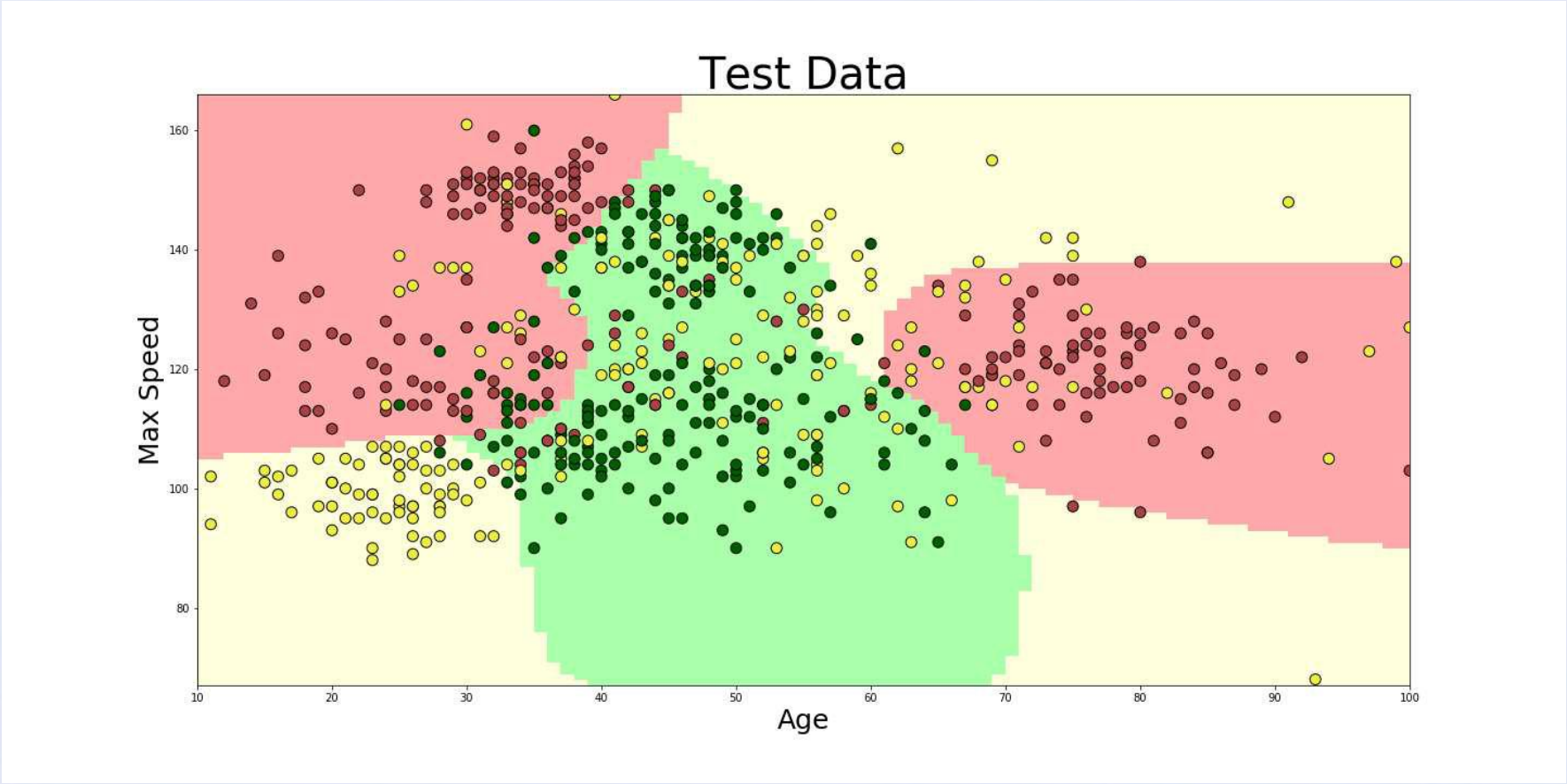


Up to 80% accuracy on test data using a regularized neural networks

Part III

Evaluation

Great results



BUT ARE THEY?

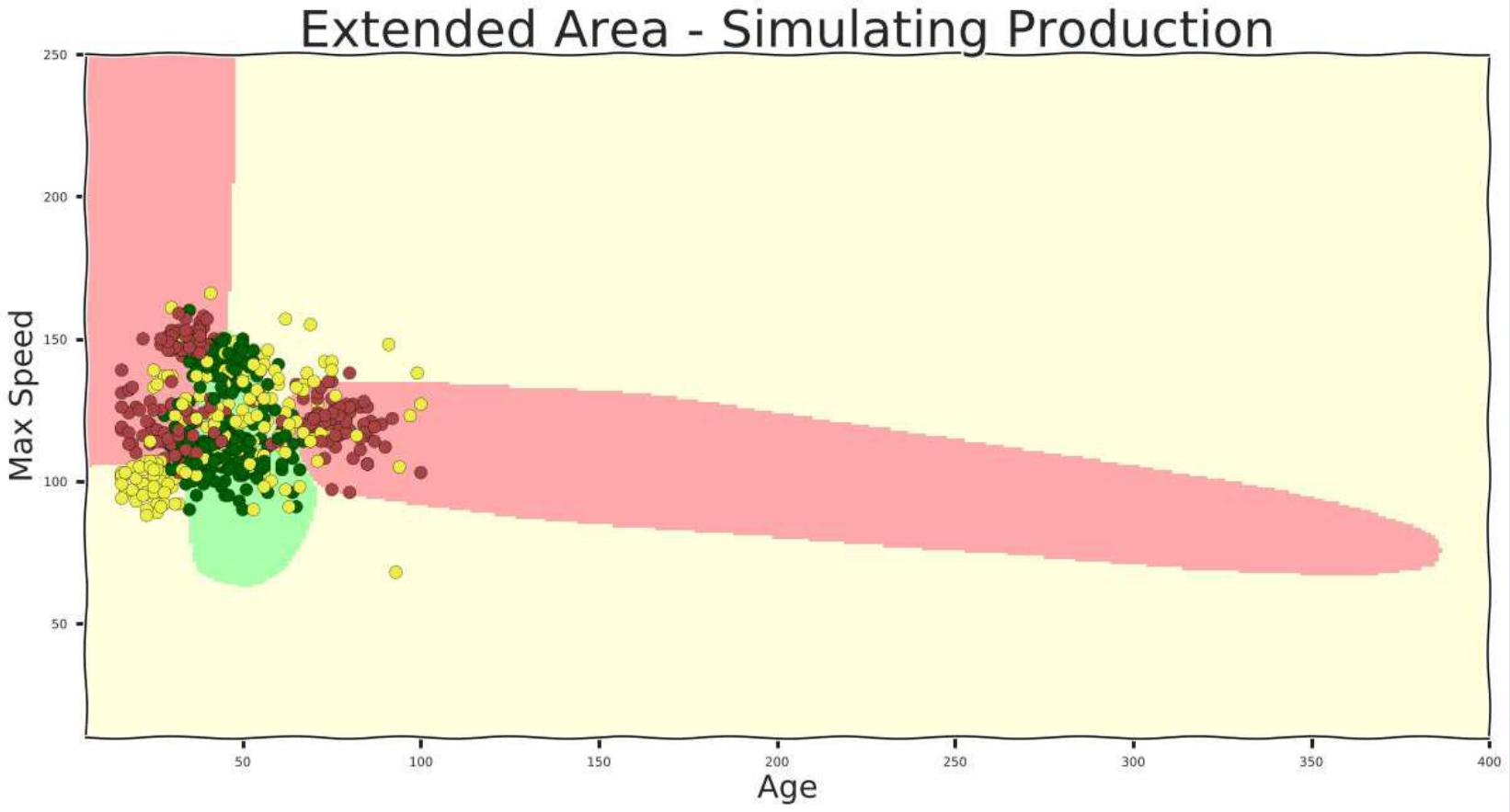
Predictions outside of the trained range

A neural network deployed in the wild may be asked to make predictions for inputs that were drawn from a different distribution than that of the training data.

A plethora of work has demonstrated that it is easy to find or synthesize inputs for which a neural network is highly confident yet wrong.

https://arxiv.org/abs/1810.09136

Looking at areas not covered by training data



https://colab.research.google.com/github/djcordhose/ai/blob/master/notebooks/sklearn/overview.ipynb#sc

How to deal with those blank spots?

First: Be aware of the reasonable range of values

- For requests in extended area
  - add a warning
  - reject request
- Identify suspicious scores
  - in exteded area scores often are very biased towards one category (1.0 score)
- Add likelihoods to scores
  - Bayesian approaches are the new hype
- Try to get more data, even incrementally
  - Use user feedback, give it lower training score

Check for bias / unfairness

your model contains all biases and truths of the training data

- Your model might perform well for a few users, but badly for others
- Accuracy and loss only show the average
- Some areas might not have properly covered with data samples (including test)
- Curse of High Dimensions

https://developers.google.com/machine-learning/fairness-overview/

Dealing with Bias / Unfairness

- log real requests
- log matching user behavior
- evaluate those requests (Elastic)
- if you recommend, do users follow your advice?
- if there is probabilty, how good is it?
- do you see bias/unfairness?
- put more emphasis on data that suffers from bias

Always check with GDPR

Explaining your models

LIME:

https://github.com/marcotcr/lime/blob/master/README.md

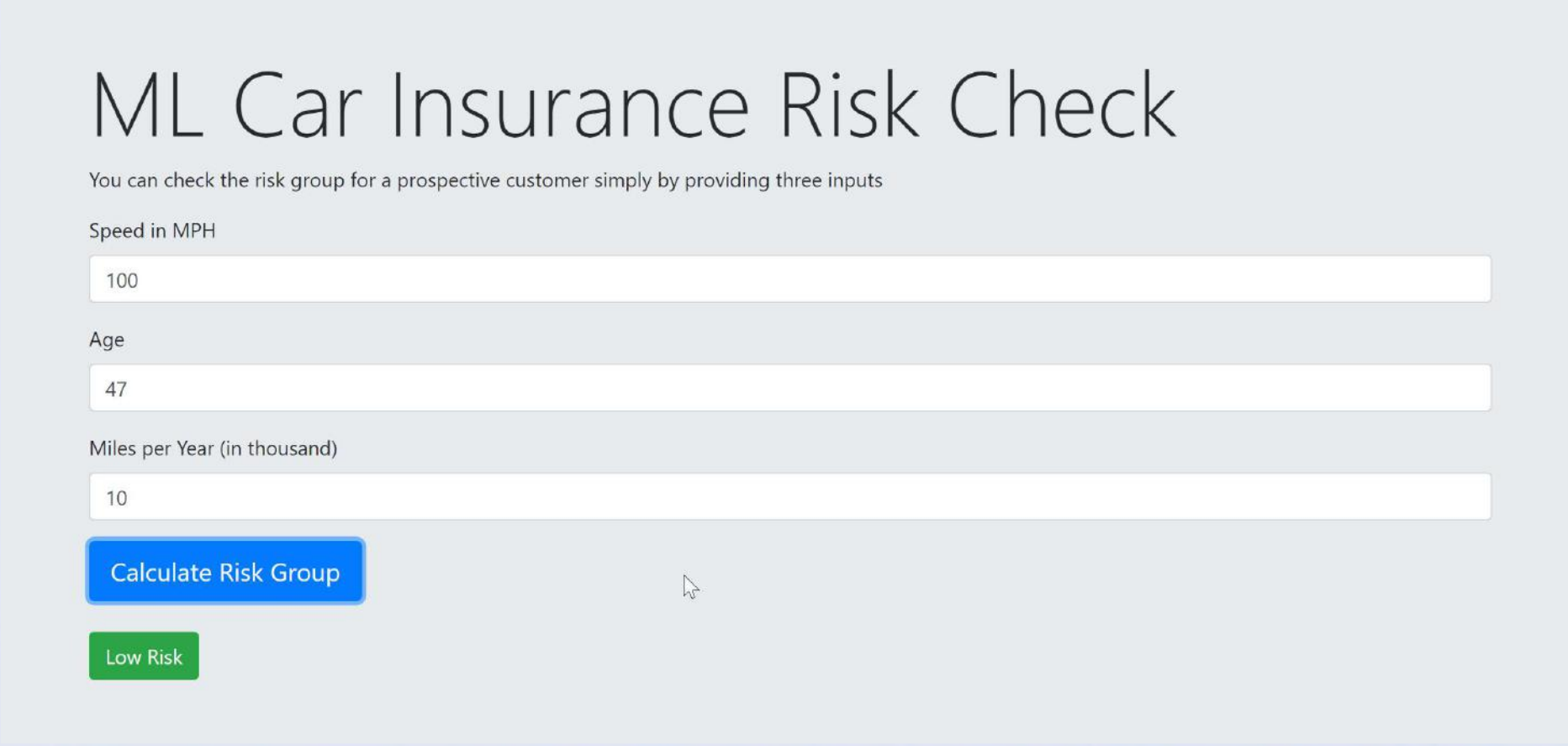
Anchor:

https://github.com/marcotcr/anchor/blob/master/README.md

Part IV:

Production

ML Car Insurance Risk Calculator



https://djcordhose.github.io/ai/html/calculator.html

Serving Models

- Be very sure you have the same pipeline of steps when serving as you had for training and evaluation
- Log real world predictions (if ok with GDPR)

It might make sense to use: http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Same goes for TensorFlow: https://github.com/tensorflow/transform / https://beam.apache.org/

Version together

1. Code for Cleaning and Trained Model
2. Trained Model
3. Serving Code
4. Training data might not be feasible because of size or GDPR

Possible Deployment Environments

- Google Cloud ML
  - Can serve TensorFlow and Sklearn
  - your model and calls are in Google's control
  - Takes away the scaling and installation burden
- Local Server
  - TensorFlow Serving
  - Flask Server
- Browser: https://js.tensorflow.org

Part V:

Post-Production

Rich Hickey
@richhickey

Following

Compile-time and runtime are less relevant notions than before-shipping and after-shipping

4:08 PM - 15 Nov 2018

https://twitter.com/richhickey/status/1063086406980026370

Going to production changes everything

True for classic software as well as machine learning models

Every deployment of a newly trained model is like a major release in software development

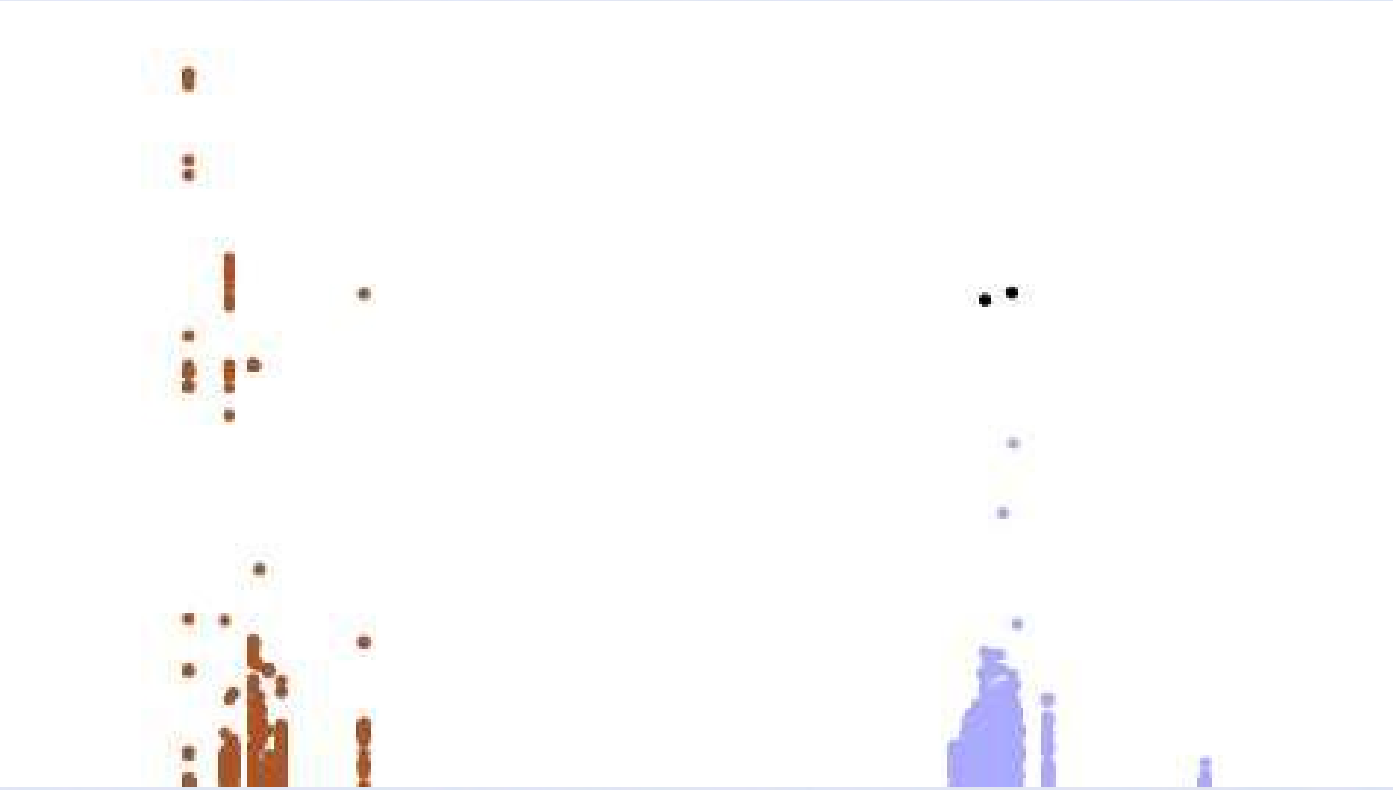
- needs to be tested like this
- or you need to make sure it rather is like a minor or bugfix release
  - make incremental changes
  - combine models

Continuity in model predictions

- People who use your model regularly will expect most things to stay the way they know it
- Even if it not totally accurate
- That might mean you can not train a model from scratch even if you have better data or a better model architecture
- Consider mixing more than one model

The Turtle Effect

If the user expects a certain pattern to be in the model, they expect it to persist



Wrap Up

Training a model is the least part

- having reasonable data is the most important part
- evaluation is more than accuracy: blank spots, bias
- going to production changes everyhing
- people expect continuity

Machine Learning from Idea to Production