

# KI Summer School

Tag 3:  
Angewandtes Machine Learning

Prof. Dr. Patrick Baier

# Dozent

Prof. Dr. Patrick Baier

<https://www.linkedin.com/in/patrickbaier/>  
[patrick.baier@h-ka.de](mailto:patrick.baier@h-ka.de)

Fachbereich: Informatik

Spezialgebiet: Maschinelles Lernen

- Maschinelles Lernen im Praxiseinsatz
- Sprach- und Audioverarbeitung
- Selbstlernende Systeme (Reinforcement Learning)



# Dozent

Prof. Dr. Patrick Baier

<https://www.linkedin.com/in/patrickbaier/>  
[patrick.baier@h-ka.de](mailto:patrick.baier@h-ka.de)

Kurz-Vita:

- seit 2020: Professor für Maschinelles Lernen, Hochschule Karlsruhe.
- 2015-2020: Data Science Lead im Bereich „Fraud Detection“, Zalando, Berlin.
- 2015-2020: Freiberuflicher Data Science Consultant und Trainer.
- Promotion Informatik, Universität Stuttgart.



# Übersicht

- Bisher haben Sie die wesentlichen Basics im Bereich Machine Learning kennengelernt.
- In diesem Kurs werden wir diese Inhalte vertiefen und uns Herausforderungen im Praxiseinsatz genauer anschauen.
- Plan für den heutigen Tag:
  - 09:00 - 11:30      Teil 1: Lösen eines ML-Problems aus der Praxis
  - 13:30 - 17:00      Teil 2: Herausforderungen in der Praxis
    - 2.1 Model-Deployment
    - 2.2 Operational Settings
    - 2.3 Model Monitoring

# Teil 1: Praxis ML-Problem

- Plan für heute morgen:
  - Besprechung ML-Problem
  - Herausgabe der Daten
  - Eigenständiges Arbeiten an den Daten, gelegentliche Catch-Ups
- Ziel: Jeder soll fähig sein, einfache ML-Modelle mit gegebenem Label zu trainieren
- Ich bin die ganze Zeit für Fragen ansprechbar, einfach Hand heben und ich komme vorbei.

# Datensatz

- Szenario: Wir betreiben einen Bücher Online-Shop und wollen beim Bestellvorgang vorhersagen ob ein Kunde seine Bestellung zurückschickt.
- Gegeben sind historische Bestelldaten in folgendem Format:

	transactionId	basket	customerType	totalAmount	returnLabel
0	7934161612	[3]	existing	77.0	0
1	5308629088	[5, 3, 0, 3]	existing	64.0	0
2	1951363325	[3, 3, 1, 4]	new	308.0	1
3	6713597713	[2]	existing	74.0	0
4	8352683669	[4, 4, 4, 4]	new	324.0	1

- Aufgabe: Bauen Sie ein ML-Modell welches das Label `returnLabel` vorhersagt.

# Datensatz

	transactionId	basket	customerType	totalAmount	returnLabel
0	7934161612	[3]	existing	77.0	0
1	5308629088	[5, 3, 0, 3]	existing	64.0	0
2	1951363325	[3, 3, 1, 4]	new	308.0	1
3	6713597713	[2]	existing	74.0	0
4	8352683669	[4, 4, 4, 4]	new	324.0	1

- transactionId : Vom unserem Online-System zufällig generierte ID der Bestellung.
- basket : Welche Bücher wurden bestellt (siehe nächste Folie).
- customerType : Ist der Kunde neu oder hat er schon bei uns bestellt?
- totalAmount : Wie hoch ist der Warenwert der Bestellung?
- returnLabel : Wurde Artikel zurückgeschickt (= 1) oder behalten (= 0).

# Datensatz

	transactionId	basket	customerType	totalAmount	returnLabel
0	7934161612	[3]	existing	77.0	0
1	5308629088	[5, 3, 0, 3]	existing	64.0	0
2	1951363325	[3, 3, 1, 4]	new	308.0	1
3	6713597713	[2]	existing	74.0	0
4	8352683669	[4, 4, 4, 4]	new	324.0	1

- Das Attribut `basket` enthält eine Liste von Bücherkategorien im Warenkorb.
- Die Werte im Basket entsprechen Bücherkategorien von 0 bis 5. (Zum Beispiel 0: Roman, 1: Science Fiction, 2: Gedichtband, usw.)
- Beispiel: `basket = [5, 3, 0, 3, 1, 1]`
  - Der Kunde hat ein Buch der Kategorie „5“, zwei Bücher der Kategorie „3“, ein Buch der Kategorie „0“ und zwei Bücher der Kategorie „1“ bestellt. In der Bestellung ist kein Buch der Kategorie „2“ und „4“.

# Datensatz

- Der Datensatz liegt in zwei Dateien vor: `train.csv` und `test.csv`
- Beide können über [GitHub](#) unter `data` runtergeladen werden.
- Die Daten in `train.csv` sollen für das Modelltraining und die Cross-Validation benutzt werden.
- Die Daten in `test.csv` werden zum Testen des Modells benutzt, um die finale Performance zu messen.
- Achtung: Das Trennzeichen („Delimiter“) in den Files ist ein Semikolon, das muss Pandas wissen, sonst schlägt das Datenladen fehl.

# Aufgabe

- Allgemeines Ziel: Es soll ein Machine-Learning Modell trainiert werden, das eine möglichst hohe Accuracy auf den Testdaten erzielt.  
Aber: Die Testdaten dürfen während des Trainings nicht benutzt werden.
- Schritte:
  1. Laden Sie die Trainingsdaten in einen Pandas DataFrame.
  2. Entfernen Sie die fehlenden Werte.
  3. Transformieren Sie die kategorischen Features mittels One-hot-encoding.
  4. Versuchen Sie auf Basis des Attributs **basket** Features zu bauen (z.B. wie oft kommt jede Kategorie im Basket vor).

# Aufgabe

- Trainieren Sie die folgenden Klassifikationsmodelle:
  1. Logistische Regression
  2. Random Forest
  3. Gradient Boosting Tree

# Aufgabe

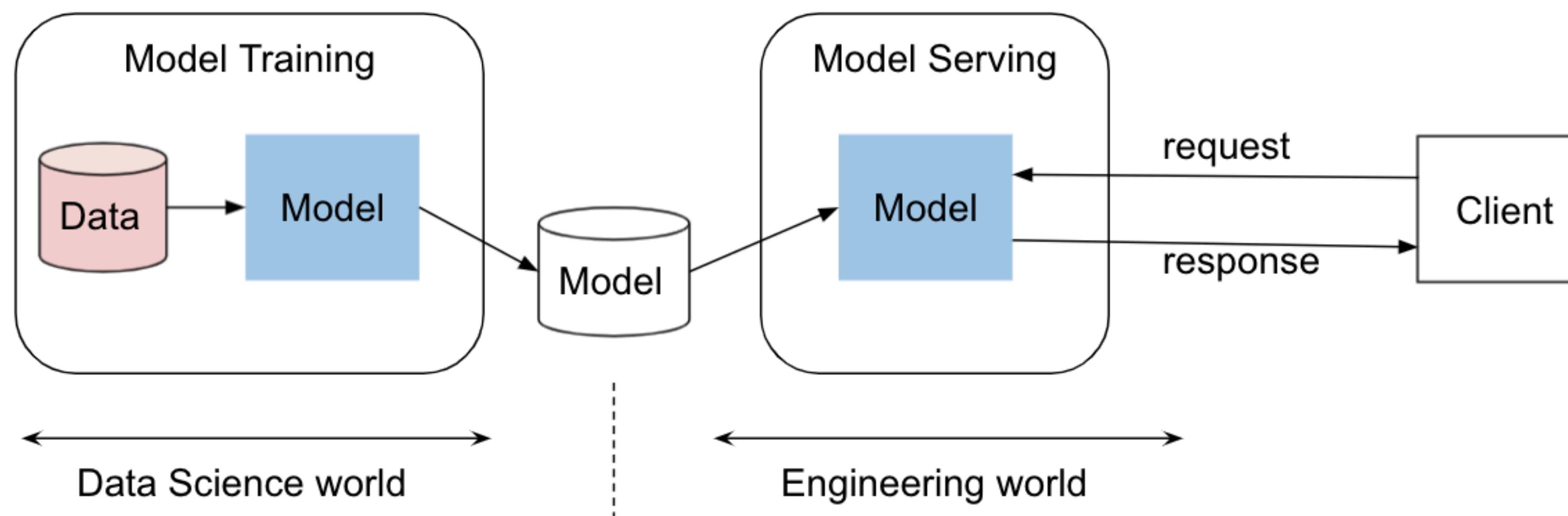
8. Laden Sie die Testdaten.
9. Entfernen Sie alle Zeilen mit fehlenden Werten.
10. Transformieren Sie die Attribute genauso wie bei den Trainingsdaten.
11. Machen Sie eine Vorhersage auf den Testdaten mit allen drei Modellen.
12. Berechnen Sie für jedes der drei Modell Accuracy, Precision und Recall.
13. Berechnen Sie außerdem die Accuracy auf den Trainingsdaten und vergleichen Sie Accuracy auf Trainings- und Testdaten. Liegt Overfitting vor?

# Herausforderungen in der Praxis

- Bisher: Modell gebaut welches die Retouren-Wahrscheinlichkeit voraussagt.
- Was wollen wir als nächstes damit machen?
  - Das Modell in unserem Online-Shop benutzen → Deployment
  - Das Modell für Entscheidungen nutzen → Operational Setting
  - Sicherstellen, dass das Modell tut was es soll → Monitoring

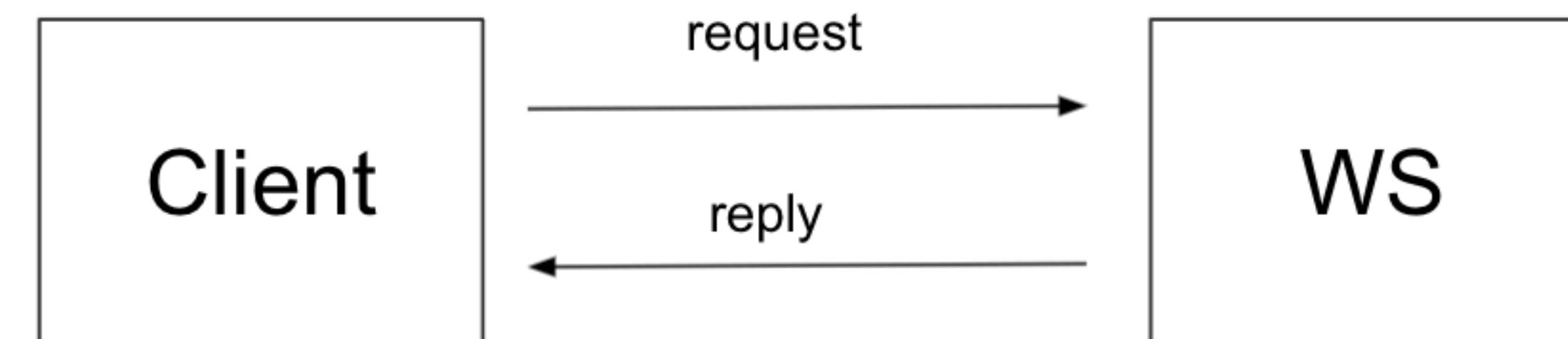
# Deployment

- Bisher läuft unser Modell nur innerhalb unseres Jupyter-Notebooks.
- Damit wir es in unserem Online-Shop nutzen können, muss es aber für das Backend-System des Shops zugänglich sein.
- D.h. wir müssen das Modell irgendwo deployen, damit ein Client anfragen zum Modell schicken kann. Das nennt man auch „Model Serving“.



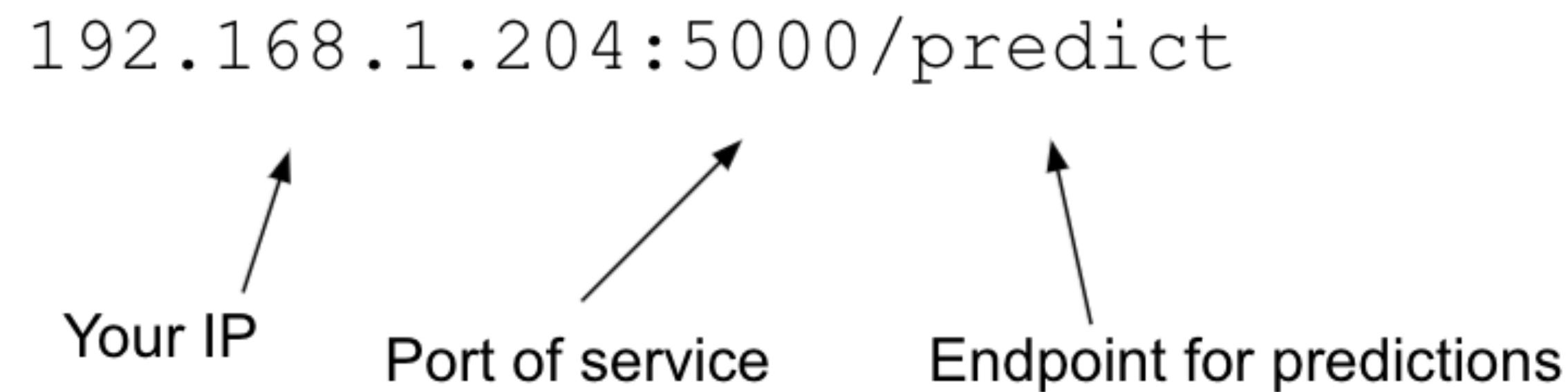
# Webservice

- Der einfachste (und oft auch beste) Weg ist das Modell in einen eigenen **Webservice** zu packen.
- Generell ist ein Webservice ein Programm welches Anfrage (Requests) von anderen Programmen (Clients) erhält, und Antworten (Responses) zurückliefert.
- Bekanntestes Beispiel: HTTP-Webserver



# Webservice

- Wir laden unser Modell in einen Webservice und machen es für Clients verfügbar.
- Um für Clients auffindbar zu sein, nutzen wir folgendes Adressschema:



# Flask-Webservice



# Flask

web development,  
one drop at a time

- Flask ist ein Python-Package welches uns schnell erlaubt einen Web-Service zu bauen.
- Das Code-Grundgerüst sieht so aus:

Bind  
method to  
predict  
endpoint  
of server

```
@app.route('/predict', methods=['POST'])
def predict():
    basket = request.json['basket']
    zipCode = request.json['zipCode']
    totalAmount = request.json['totalAmount']
    p = probability(basket, zipCode, totalAmount)
    return jsonify({'probability': p}), 201
```

Return return probability to client (as json)

Extract  
fields from  
request  
that was  
sent by  
client

# Flask-Webservice

- Wenn unser Webservice gestartet ist, können wir auf dem gleichen Computer einen Test-Request an den Client schicken.
- Dies kann z.B. einfach über das curl-Kommando erfolgen:

```
curl -i -H "Content-Type: application/json" -X POST -d '{"transactionId":  
6304965406, "basket": [4, 3, 0, 3, 1, 1, 2, 0, 2], "zipCode": 2729,  
"totalAmount": 12}' http://localhost:5000/predict
```

target address

request data

## Aufgabe 2:

1. Erweitern Sie den Flask-Server so, dass er den Test-Request mit Hilfe des ML-Modells beantwortet (momentan wird immer 0.0 zurückgegeben).
  - Speichern Sie am Ende des Trainings das trainierte Modell auf die Festplatte:  
[http://scikit-learn.org/stable/modules/model\\_persistence.html](http://scikit-learn.org/stable/modules/model_persistence.html)
  - Laden Sie das Modell beim Starten des Servers
  - Leiten Sie die Daten des Test-Requests in das Modell
  - Geben Sie die Vorhersagewahrscheinlichkeit des Modells zurück an den Client.
2. Führen Sie die Datei `loadSimulator.py` aus. Dadurch werden mehrere Requests an den Server geschickt. Finden Sie heraus, warum einige der Request fehlschlagen.

# Fehlende Daten

- Der LoadSimulator schlägt fehlt, da in den Requests die er sendet ab und zu ein Datenpunkt fehlt.
- Wie geht man damit um?
  - Einfach keine Antwort schicken? Oft keine gute Option!
  - Datenpunkt auffüllen? Gut, aber mit welchem Wert?
  - Datenpunkt mit möglichst neutralem Wert auffüllen!
  - Welcher Wert ist neutral?
    - Durchschnitt (numerisches Feature) oder häufigster Wert (kategorisches Feature) der Trainingsdaten. (Wie bei den Trainingsdaten schon praktiziert).

# Fehlende Daten

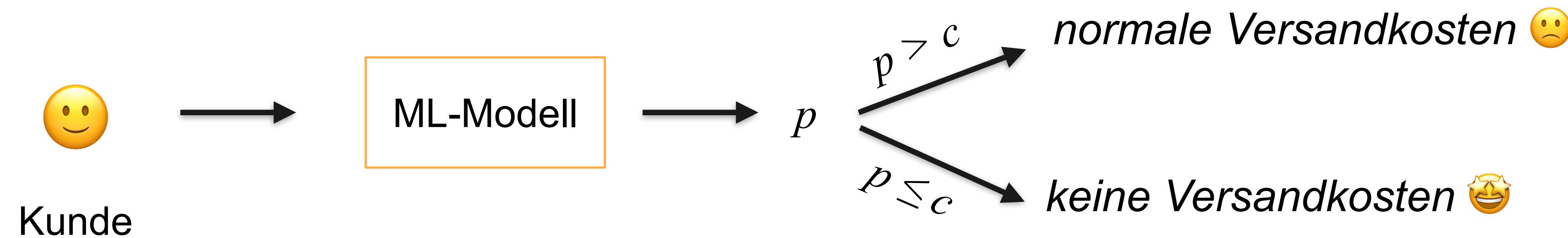
- Fehlenden Werte mit Durchschnitt (numerisches Feature) oder häufigster Wert (kategorisches Feature) der Trainingsdaten. (Wie bei den Trainingsdaten schon praktiziert).
- Frage: Wie weiß der Server, welches der durchschnittliche Wert von Feature A in den Trainingsdaten war?
- Lösung: Wir speichern uns diese Werte während des Trainings und laden Sie in den Flask-Server (genauso wie das Modell selbst).

## Aufgabe 3:

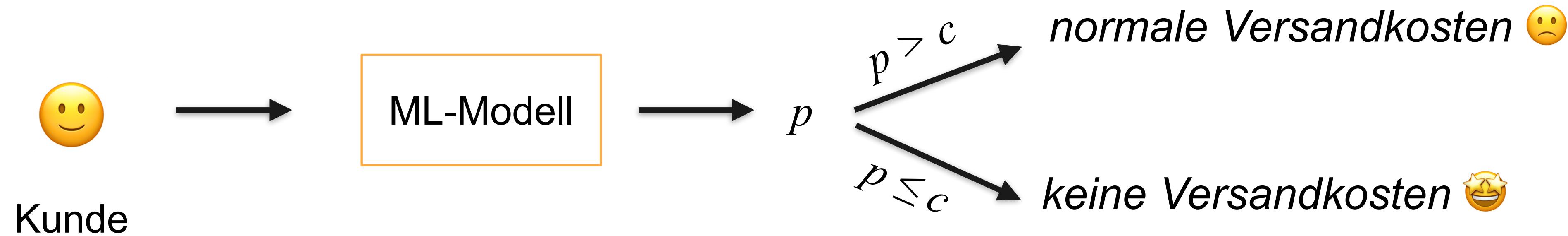
1. Speichern Sie am Ende des Trainingsnotebooks für jedes Feature den Durchschnittswert in ein File (z.B als Dictionary).
2. Laden Sie dieses File in den Flask-Server.
3. Checken Sie für jeden eingehenden Request ob das Feature da ist, und wenn nein, füllen Sie das Feature mit dem Durchschnittswert aus dem Training auf.

# Operational Setting

- Modelle zur binären Klassifizierung (Label/Klasse ist entweder 0 oder 1), geben in der Regel eine Wahrscheinlichkeit  $p \in [0,1]$  als Vorhersage zurück. Dies ist die Wahrscheinlichkeit, dass der Datenpunkt zu Klasse 1 gehört.
  - Hohe Wahrscheinlichkeit: Modell ist sicher dass Datenpunkt zu Klasse 1 gehört.
  - Niedrige Wahrscheinlichkeit: Modell ist sicher dass Datenpunkt zu Klasse 0 gehört.
- Nehme wir folgendes an: Wir wollen das Modell benutzen um zu entscheiden, ob Kunde beim Online-Shopping kostenlose Versandkosten angeboten bekommt:



# Operational Setting



- Wieso sollten wir sowas machen?  
Die Retoure-Kosten die wir bei Kunden mit hoher Rücksendewahrscheinlichkeit haben, versuchen wir durch Versandkosten zu kompensieren.
- Bei welchem Grenzwert  $c$  entscheiden wir, ob wir Versandkosten verlangen oder nicht?
- Diese Entscheidung nennen wir *Operational Setting* und hat erhebliche Auswirkungen auf unser Business!

# Exkurs: Model Evaluation

- Mithilfe des Grenzwerts  $c$  wird eine Wahrscheinlichkeit  $p$  in eine Vorhersage 0 oder 1 umgewandelt.
- Standardmäßig ist dieser Grenzwert oft auf  $c = 0.5$  gesetzt, was aber nicht immer sinnvoll, wie wir später sehen werden.
- Wenn wir für einen Datenpunkt die Vorhersage (Prediction) mit dem tatsächlichen Label (True label) vergleichen, tritt einer der vier Fälle rechts ein.
- Aus diesen Fällen lassen sich verschiedene Metriken ableiten, z.B. Accuracy, Precision und Recall.

		True label	
		one	zero
Prediction	one	True positive	False positive
	zero	False negative	True negative

Confusion Matrix

# Exkurs: Model Evaluation

Bei der Evaluierung wird für jeden Datenpunkt in den Testdaten bestimmt, welcher der Entscheidungsfall eintritt.

Beispiel:

Vorhersage	Label	Fall
1	1	tp
1	0	fp
0	1	fn
1	0	fp
1	1	tp
0	1	fn
0	1	fn
0	0	tn

- 8 Testdatenpunkte:
- 2 true positives
  - 2 false positives
  - 3 false negatives
  - 1 true negative

# Exkurs: Accuracy

Was macht ein gutes Modell aus?

1. Die Vorhersagen sollen möglichst **genau** sein:

- Gute Vorhersagen: True positives oder true negatives.
- Schlechte Vorhersagen: False positives oder false negatives.
- Aus dieser Überlegung leitet sich die Metrik „**Accuracy**“ ab:

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{all}}$$

Bedeutung: “Wieviel Prozent der Vorhersagen sind korrekt?”

# Exkurs: Precision

Was macht ein gutes Modell aus?

2. Die Vorhersagen sollen möglichst **präzise** sein:

- Wenn das Modell eine „1“ vorhersagt, dann soll es auch wirklich eine „1“ sein.
- Um dies zu messen, nutzen wir die Metrik „**Precision**“:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Bedeutung: “Wieviel Prozent der vorhergesagten positiven Fälle („1“) sind auch wirklich positiv?“.

# Exkurs: Recall

Was macht ein gutes Modell aus?

3. Die Vorhersagen sollen eine **hohe Trefferquote** haben:

- Das Modell soll möglichst alle „1“-Fälle in den Testdaten als solche erkennen.
- Um dies zu messen, nutzen wir die Metrik „**Recall**“:

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Bedeutung: “Wieviel Prozent der tatsächlich positiven Fälle („1“) erkennt das Modell?“.

# Exkurs: F1 Score

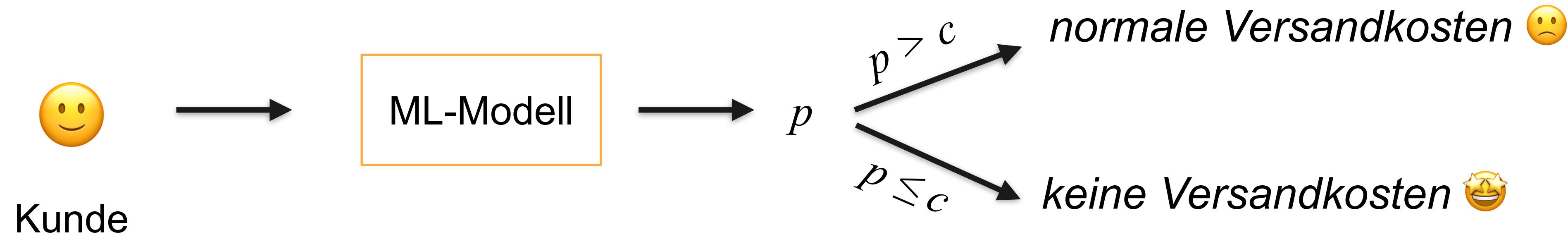
Was macht ein gutes Modell aus?

4. Die Vorhersagen sollen sowohl eine **hohe Trefferquote** haben als auch **präzise** sein:

- Kombination von precision und recall.
- Um dies zu messen, nutzen wir die Metrik „**F1 score**“:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

# Operational Setting



- Nehmen wir an, dass der Grenzwert  $c$  ziemlich ...
  - hoch gewählt ist: Nur wenn  $p$  sehr hoch ist, gibt es normale Versandkosten.  
D.h. es gibt weniger *false positives* aber mehr *false negatives*.
  - niedrig gewählt ist: Nur wenn  $p$  sehr niedrig ist, gibt es keine Versandkosten.  
D.h. es gibt weniger *false negatives* aber mehr *false positives*.
- Beobachtung: Wir können durch variieren des Grenzwerts  $c$  unterschiedliche Fehlerfehler reduzieren

# Operational Setting

- Beispiel: Gegeben sind fünf Datenpunkte mit der Vorhersagewahrscheinlichkeit des Modells und dem tatsächlichen Label.
- Wenn der Grenzwert  $c = 0.5$  gewählt wird ergibt sich folgendes Bild:

	Prediction	True label
class one	0.9	1
	0.8	1
class zero	0.4	0
	0.2	1
	0.1	0

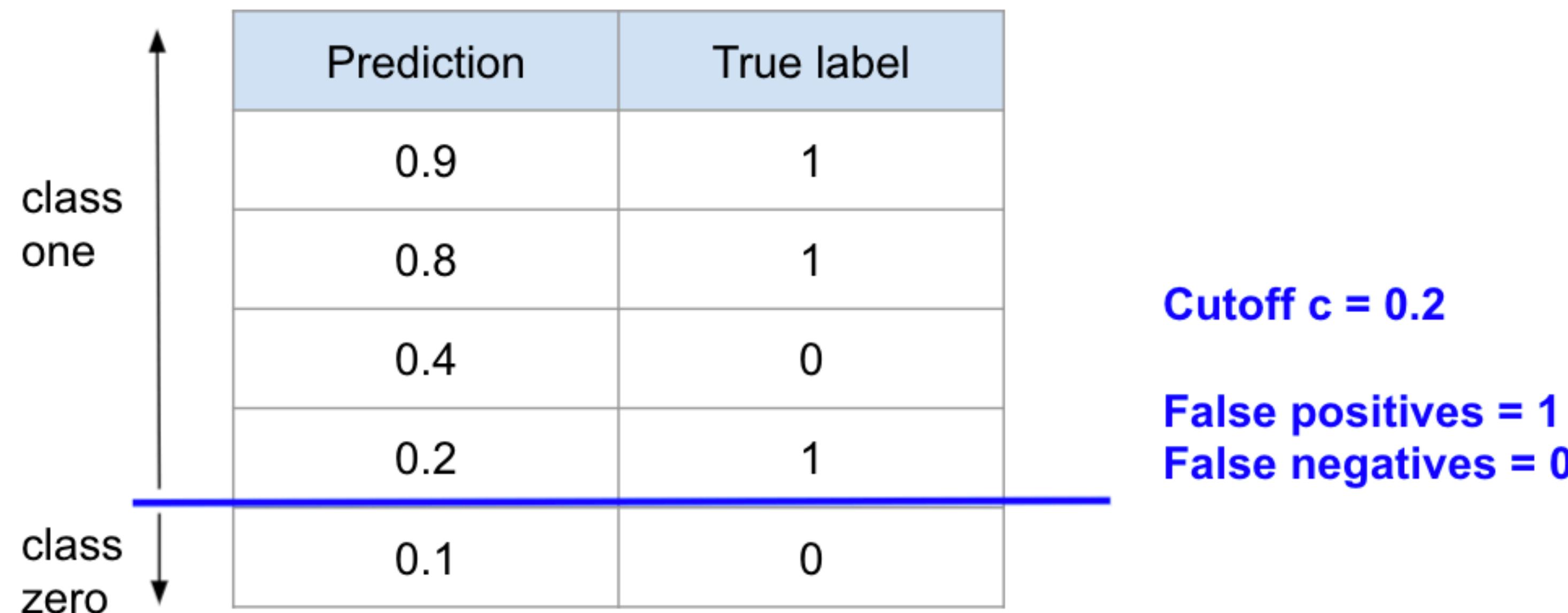
Cutoff  $c = 0.5$

False positives = 0

False negatives = 1

# Operational Setting

- Ändert man den Grenzwert auf  $c = 0.2$  ändern sich die Fehlertypen:



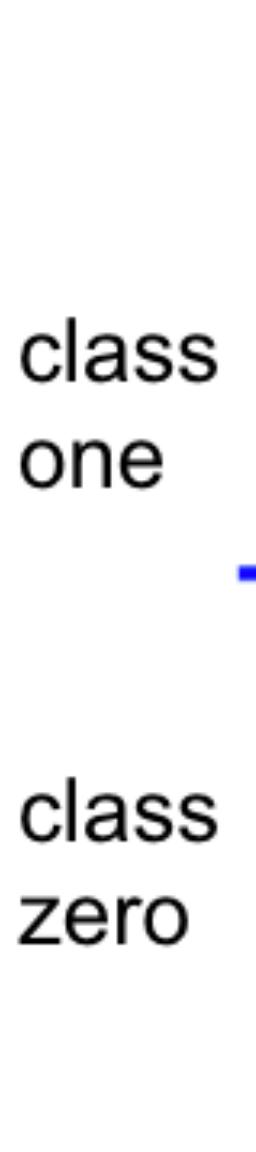
- Frage: Welches ist für mein Business-Case der beste Grenzwert?

# Operational Setting

- Frage: Welches ist für mein Business-Case der beste Grenzwert?
- Antwort: Es kommt auf die Kosten der einzelnen Fälle an!
- Beispiel anhand des Online-Shops:
  - **true positive**: Normale Versandkosten und Kunde hätte retourniert  
→ Wenn Kunde abspringt (nicht bestellt): Kosten für Bearbeitung gespart, im Schnitt ca. +5€
  - **false positive**: Normale Versandkosten und Kunde hätte nicht retourniert.  
→ Wenn Kunde abspringt (nicht bestellt): Bestellung entgangen, im Schnitt ca. -50€
  - **true negative**: Keine Versandkosten und Kunde hätte nicht retourniert.  
→ Wenn Kunde nicht abspringt (bestellt): An Bestellung verdient, im Schnitt ca. + 50€
  - **false negative**: Keine Versandkosten und Kunde hätte retourniert.  
→ Wenn Kunde nicht abspringt (bestellt): Kosten für Bearbeitung, im Schnitt ca. -5€

# Operational Setting

- Gegeben unserer Testdatensatz können wir jetzt für verschiedene Grenzwerte  $c$  ausrechnen wie hoch der Gewinn/Verlust ist.
- Beispiel für  $c = 0.5$  :



	Prediction	True label
class one	0.9	1
	0.8	1
class zero	0.4	0
	0.2	1
	0.1	0

- 2 true positives a 5€
- 0 false positives a - 50€
- 2 true negatives a +50€
- 1 false negatives a -5€

---

Gewinn: 115 €

# Operational Setting

- Gegeben unserer Testdatensatz können wir jetzt für verschiedene Grenzwerte  $c$  ausrechnen wie hoch der Gewinn/Verlust ist.
- Beispiel für  $c = 0.2$  :



	Prediction	True label
class one	0.9	1
	0.8	1
	0.4	0
	0.2	1
class zero	0.1	0

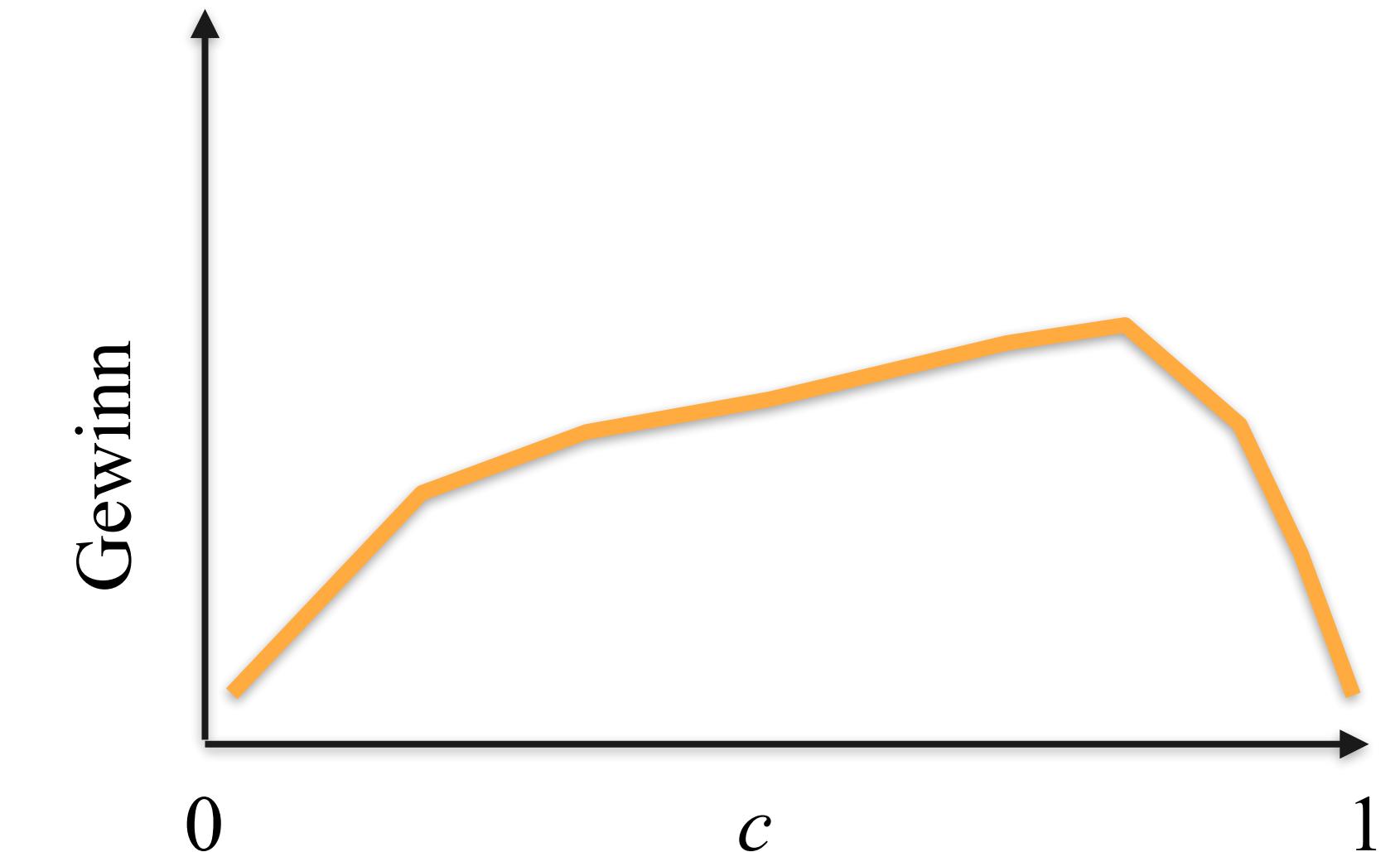
- 3 true positives a 5€
- 1 false positives a - 50€
- 1 true negatives a +50€
- 0 false negatives a -5€

---

Gewinn: 15 €

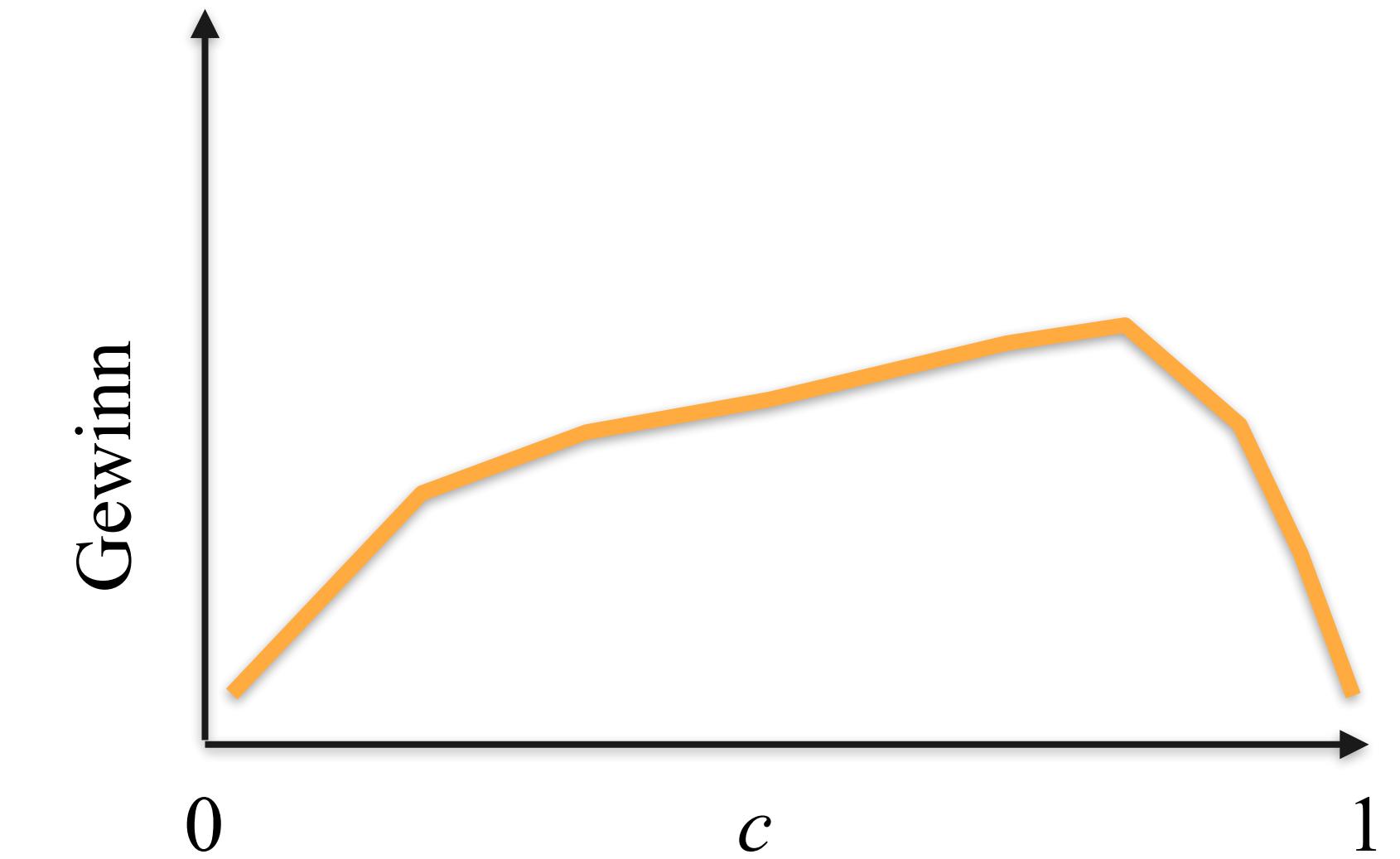
# Operational Setting

- Gegeben unserer Testdatensatz können wir jetzt für verschiedene Grenzwerte  $c$  ausrechnen wie hoch der Gewinn/Verlust ist.
- Wenn wir das für jeden möglichen Grenzwert durchtesten können wir den Plot rechts erstellen, aus dem sich gute Grenzwerte leicht ablesen lassen.
- Aber: Nicht immer ist der Gewinnmaximale Grenzwert momentan das richtige Operational Setting, da Nebeneffekte auch mit reinspielen können, z.B.  
*„Unser Lager ist zu voll, wir müssen möglichst schnell viel verkaufen“*  
→ In diesem Fall kann ein sehr hoher Grenzwert kurzfristig die Verkaufszahlen erhöhen (auf Kosten erhöhter Retourenquoten)



# Aufgabe:

- Nutzen Sie die Testdaten um einen Plot für Grenzwert und Gewinn zu erstellen.
- Die einzelnen Schritte dazu sind:
  1. Erstellen Sie einen neuen DataFrame der für jeden Datenpunkt in den Testdaten die Vorhersagewahrscheinlichkeit und das wahre Label enthält.
  2. Sortieren Sie diesen DataFrame absteigend nach den Vorhersagewahrscheinlichkeit.
  3. Bestimmen Sie für jede Wahrscheinlichkeit wieviele Fälle (tps, fps, tns, fns) es gibt und wie hoch der Gewinn wäre, wenn der Grenzwert genau auf dieser Wahrscheinlichkeit liegen würden.



# Model Monitoring

- Wenn das Model deployed wurde, müssen wir dafür sorgen dass es dauerhaft das richtige tut, d.h. gute Vorhersagen macht.
- Monitoring kennen wir von aus dem *klassischen Softwareengineering*, wo es darum geht die *technische* Funktionalität des Systems zu überwachen.
- Beispiel: Grafana Dashboard mit
  - CPU Auslastung
  - Speicherfüllstand
  - Disk-, Netzwerk-I/O
  - Durchsatz, Latency
  - ....



# Model Monitoring

- Für den Server auf dem das ML-Modell läuft müssen wir das gleich tun, haben aber noch eine zusätzliche Dimension: Sicherstellen dass die *semantische Funktionalität* gewährleistet ist, d.h. sicherstellen das die Vorhersagen gut sind.
- Dabei gilt: ML-Modelle verhalten sich nur wie erwartet, wenn die Verteilung der Eingabedaten im Live-Betrieb sehr ähnlich zu der Verteilung der Trainingsdaten ist.
- Daraus leitet sich ab: Wir müssen beim Monitoring überprüfen ob diese beiden Verteilung noch ähnlich sind.
- Wenn dies nicht der Fall ist, können die Vorhersagen des Modells nicht mehr gut sein, ohne das wir es direkt merken (→ *hoher finanzieller Schaden möglich*).

## Beispiel:

## SCENARIO

Let's deploy a model for fraud detection in an online shop!

Steps we take:

1. Collect training data.
2. Train a model.
3. Deploy it to production.

# COLLECT DATA

```
209.160.24.63 - - [03/Mar/2016:18:22:16] "GET /product.screen?productId=W-SH-A02&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 3878 "http://www.google.com" "Mozilla/5.0 (Windows NT 6.1; i
209.160.24.63 - - [03/Mar/2016:18:22:16] "GET /oldLink?itemId=EST-6&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 1748 "http://www.exploratorystore.io/oldLink?itemId=EST-6" "Mozilla/5.0 (i
209.160.24.63 - - [03/Mar/2016:18:22:17] "GET /product.screen?productId=B5-AG-G09&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 2558 "http://www.exploratorystore.io/product.screen?product
209.160.24.63 - - [03/Mar/2016:18:22:19] "POST /category.screen?categoryId=STRATEGY&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 407 "http://www.exploratorystore.io/cart.doAction=remov
209.160.24.63 - - [03/Mar/2016:18:22:20] "GET /product.screen?productId=F5-SG-G02&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 2847 "http://www.exploratorystore.io/category.screen?category
209.160.24.63 - - [03/Mar/2016:18:22:20] "POST /cart.doAction=addtocart&itemId=EST-216&productId=F5-SG-G02&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 1281 "http://www.exploratorystore.
209.160.24.63 - - [03/Mar/2016:18:22:21] "POST /cart.doAction=purchase&itemId=EST-216&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 485 "http://www.exploratorystore.io/cart.doAction=addt
209.160.24.63 - - [03/Mar/2016:18:22:22] "POST /cart.success.do?SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 3288 "http://www.exploratorystore.io/cart.doAction=purchase&itemId=EST-21"
209.160.24.63 - - [03/Mar/2016:18:22:21] "GET /cart.doAction=remove&itemId=W-SH-A01&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 3619 "http://www.exploratorystore.io/o
209.160.24.63 - - [03/Mar/2016:18:22:22] "GET /oldLink?itemId=EST-14G&SESSIONID=5085L6FF7ADFF4953 HTTP 1.1" 200 1352 "http://www.exploratorystore.io/cart.doAction=addtocart&itemId=EST-
112.111.162.4 - - [03/Mar/2016:18:26:36] "GET /product.screen?productId=W-SH-C04&SESSIONID=5075L8FF5ADF4964 HTTP 1.1" 200 778 "http://www.exploratorystore.io/category.screen?category
112.111.162.4 - - [03/Mar/2016:18:26:37] "POST /cart.doAction=addtocart&itemId=EST-186&productId=W-SH-C04&SESSIONID=5075L8FF5ADF4964 HTTP 1.1" 200 215 "http://www.exploratorystore.
112.111.162.4 - - [03/Mar/2016:18:26:38] "POST /cart.doAction=addItemId=EST-186&productId=5075L8FF5ADF4964 HTTP 1.1" 200 122 "http://www.exploratorystore.io/cart.doAction=add
112.111.162.4 - - [03/Mar/2016:18:26:38] "POST /cart.error.do?msg=CreditDoesNotMatch&SESSIONID=5075L8FF5ADF4964 HTTP 1.1" 200 1232 "http://www.exploratorystore.io/cart.doAction=purc
112.111.162.4 - - [03/Mar/2016:18:26:37] "GET /category.screen?categoryId=NULL&SESSIONID=5075L8FF5ADF4964 HTTP 1.1" 505 2445 "http://www.exploratorystore.io/category.screen?category
112.111.162.4 - - [03/Mar/2016:18:26:38] "GET /oldLink?itemId=EST-76&SESSIONID=5075L8FF5ADF4964 HTTP 1.1" 501 207 "http://www.exploratorystore.io/category.screen?category
74.125.19.106 - - [03/Mar/2016:18:32:15] "GET /cart.doAction=addtocart&itemId=EST-166&productId=DC-SG-G02&SESSIONID=5045L7FF10ADFF4998 HTTP 1.1" 200 1425 "http://www.exploratorystore.
74.125.19.106 - - [03/Mar/2016:18:32:15] "GET /category.screen?categoryId=NULL&SESSIONID=5045L7FF10ADFF4998 HTTP 1.1" 503 2039 "http://www.exploratorystore.io/oldLink?item
117.21.246.164 - - [03/Mar/2016:18:36:02] "POST /cart.doAction=changeQuantity&itemId=EST-216&productId=W-SH-A01&SESSIONID=5095L6FF80DFP5015 HTTP 1.1" 200 889 "http://www.exploratory
117.21.246.164 - - [03/Mar/2016:18:36:03] "POST /cart.doAction=addtocart&itemId=EST-276&productId=DC-SG-G02&SESSIONID=5095L6FF80DFP5015 HTTP 1.1" 200 1291 "http://www.exploratorystore.
```

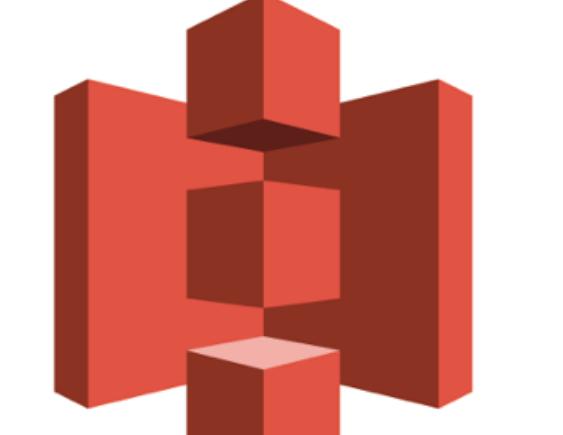
Log data



Database



Data Warehouse



Amazon S3

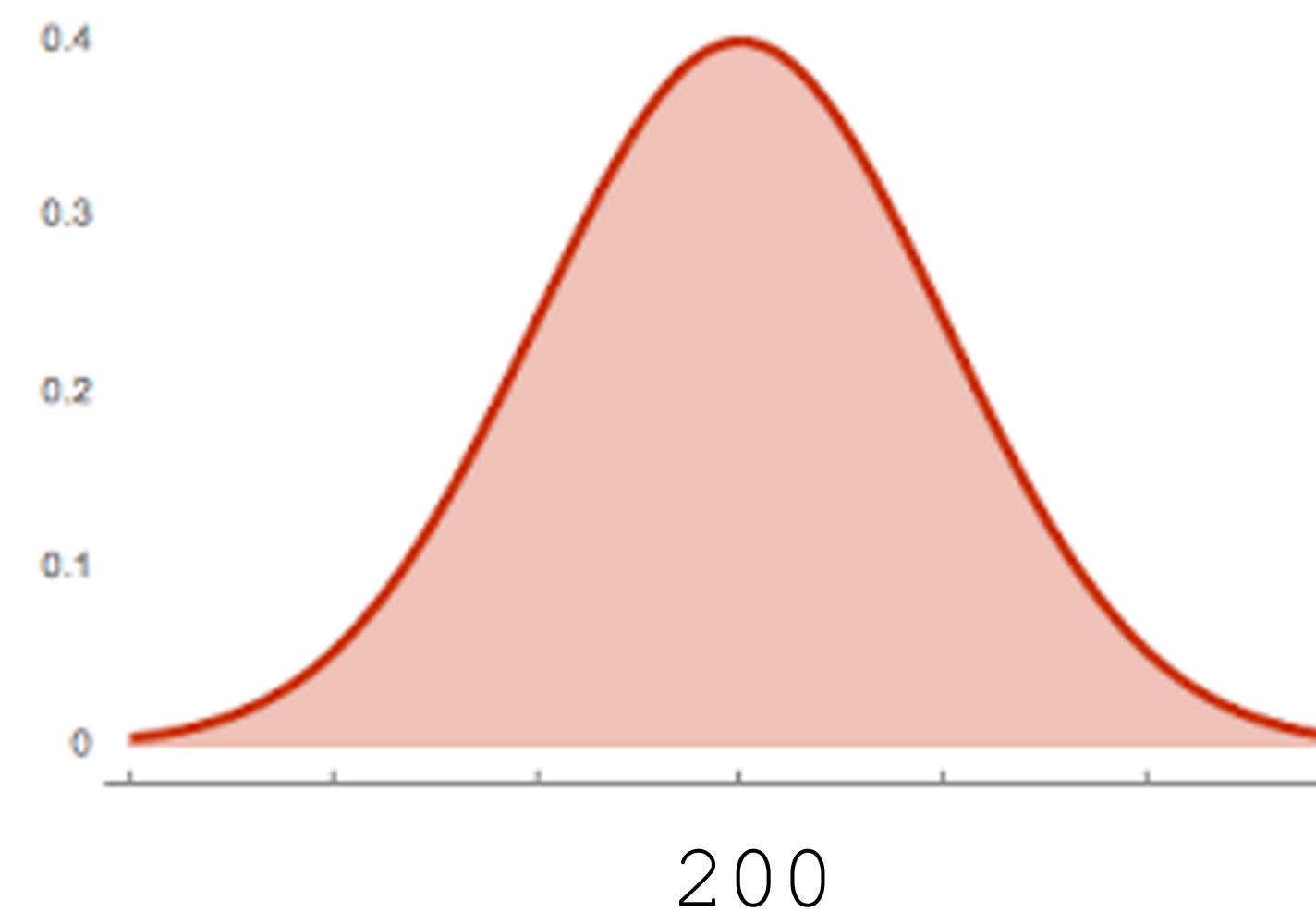
Go through the systems and collect data for training

# TRAINING DATA

#	Feature-1	Time-to-order [s]	...	Feature-N	Label
1	2	300	...	1	<b>not-fraud</b>
2	1	5	...	0	<b>fraud</b>
3	3	120	...	0	<b>not-fraud</b>
4	2	200	...	1	<b>not-fraud</b>
5	1	250	...	0	<b>fraud</b>
...	...	...	...	...	...

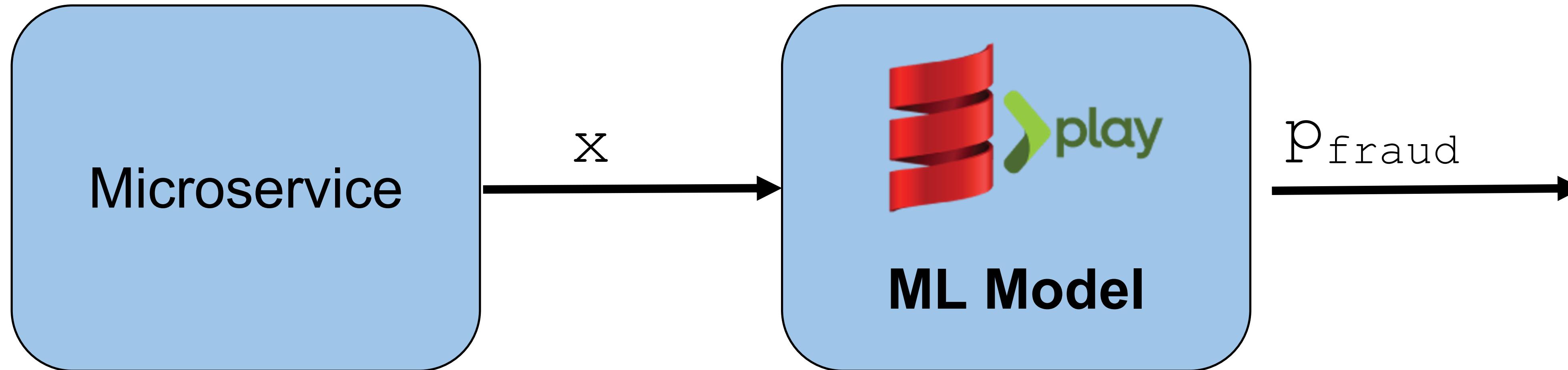
# FEATURE DISTRIBUTION

Time-to-order [s]
300
5
120
200
250
...



Distribution of feature in training data

# GO LIVE



Once we are live, we get features  $x$  sent over by a different microservice in real-time.

# MONITORING



- ✓ CPU usage
- ✓ Memory usage
- ✓ Latency
- ....

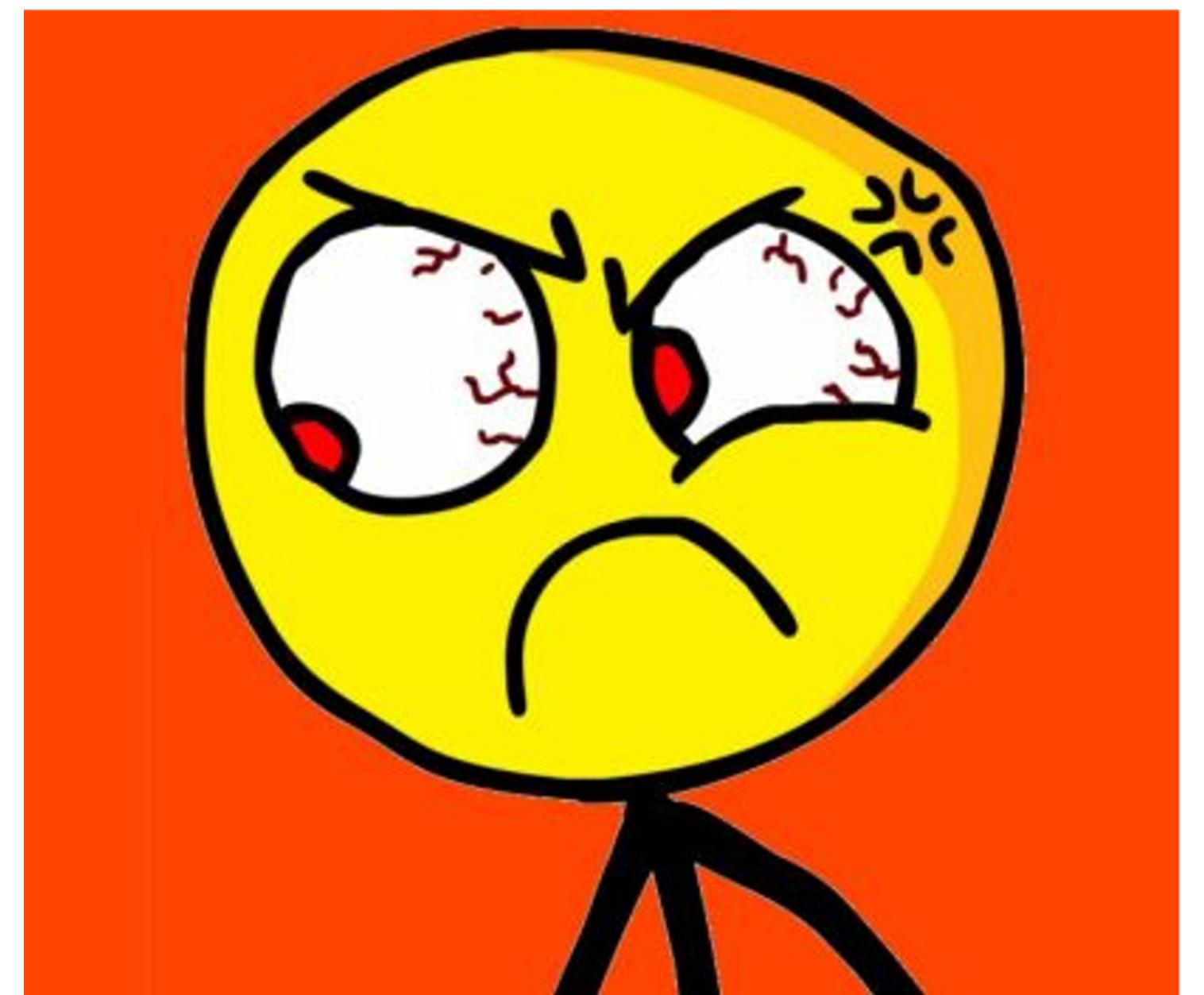
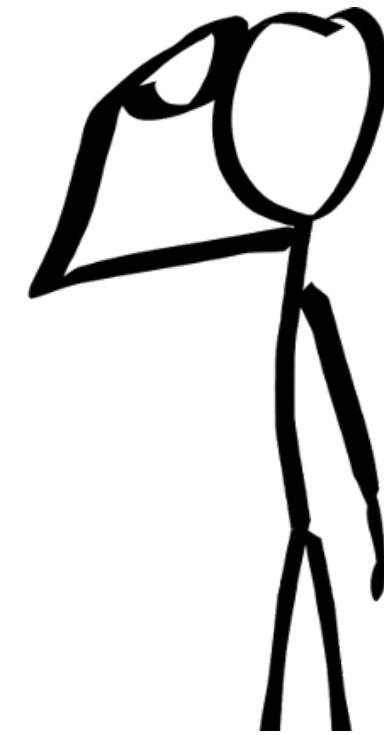


# CHANGE OF MOODS

Some weeks later, people are angry:

“We fail to detect fraud, our business is ruined!”

What happened?

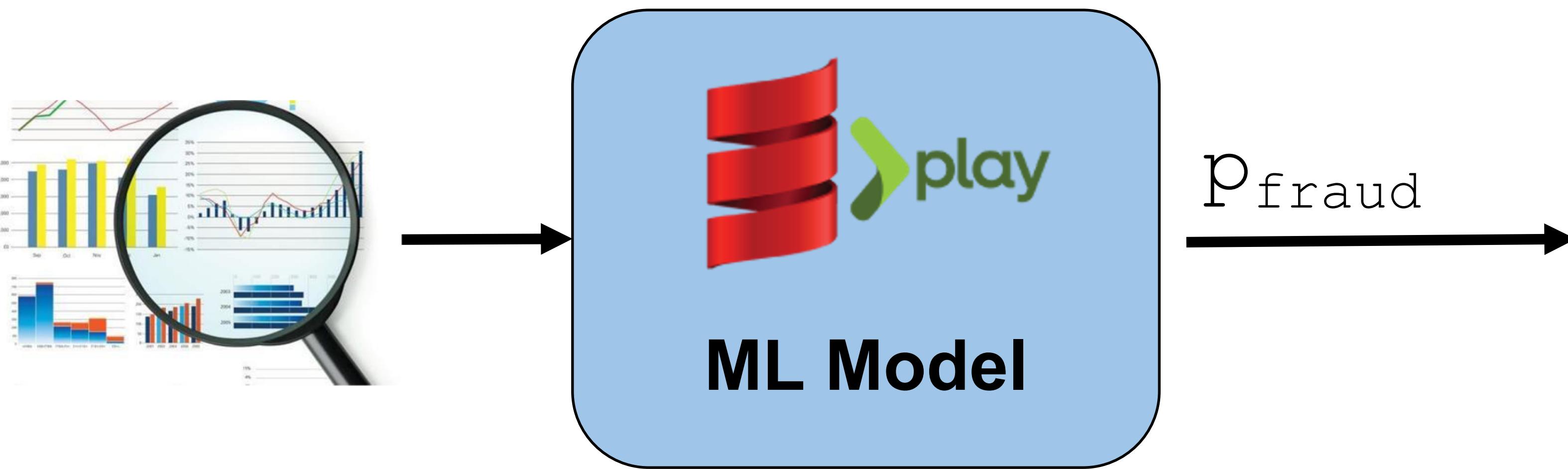


# INVESTIGATION

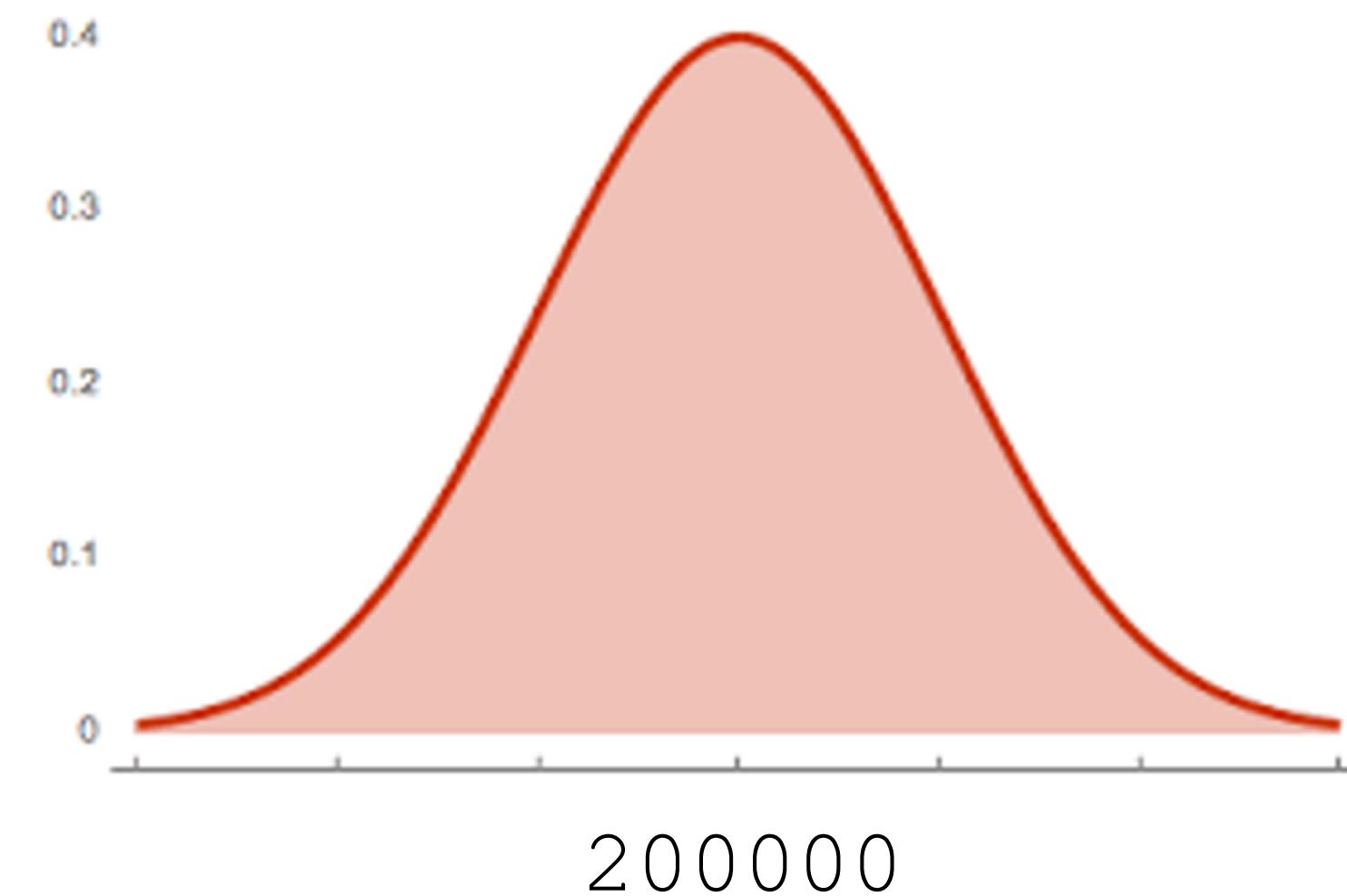


- ✓ CPU usage
- ✓ Memory usage
- ✓ Latency
- ....

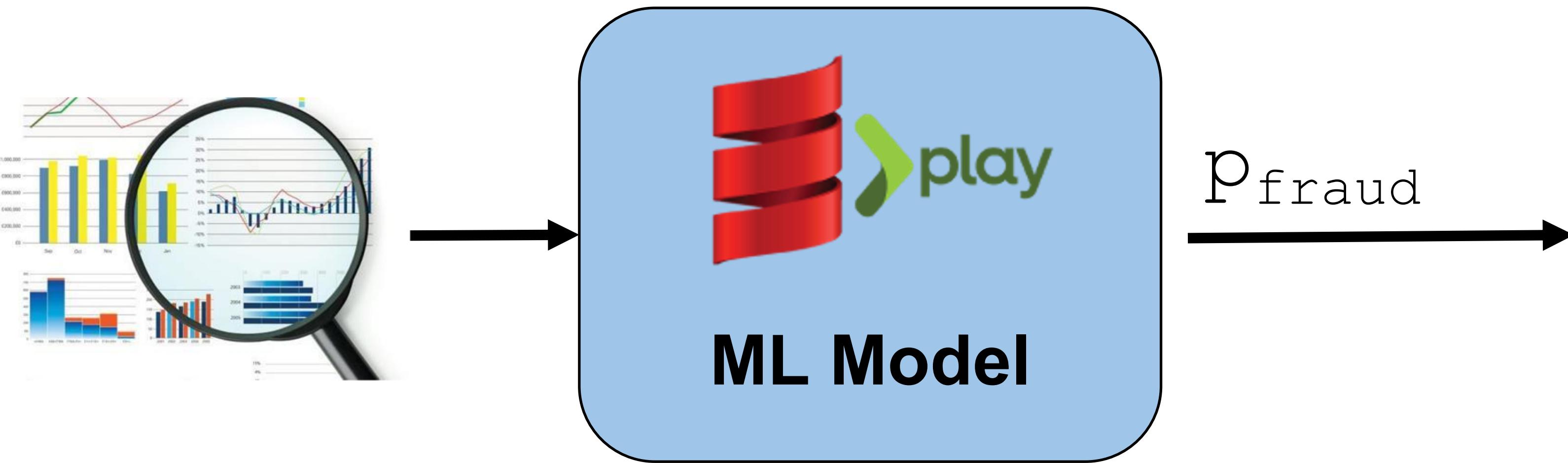
# INVESTIGATION



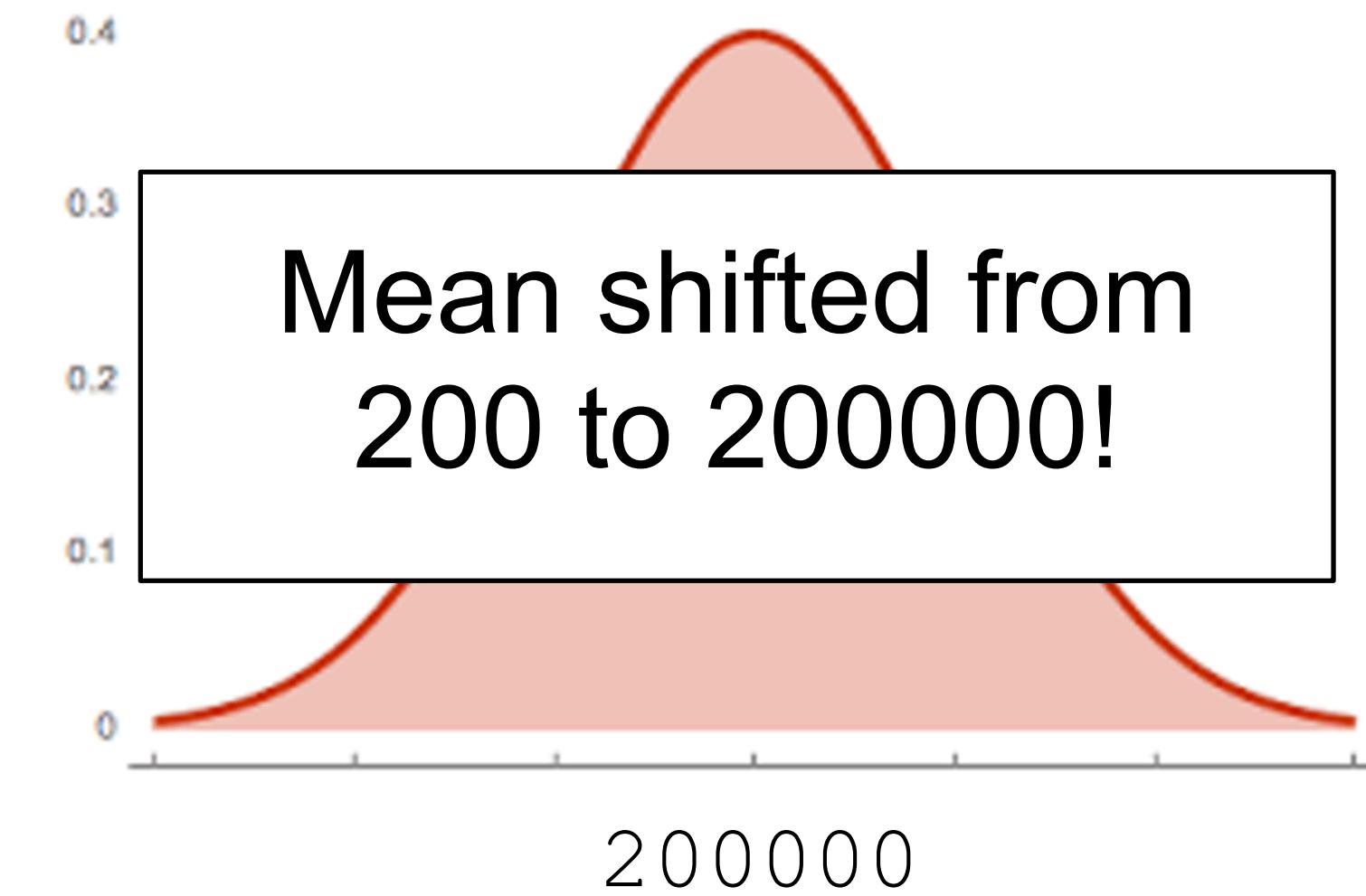
Time-to-order
300000
5000
120000
...



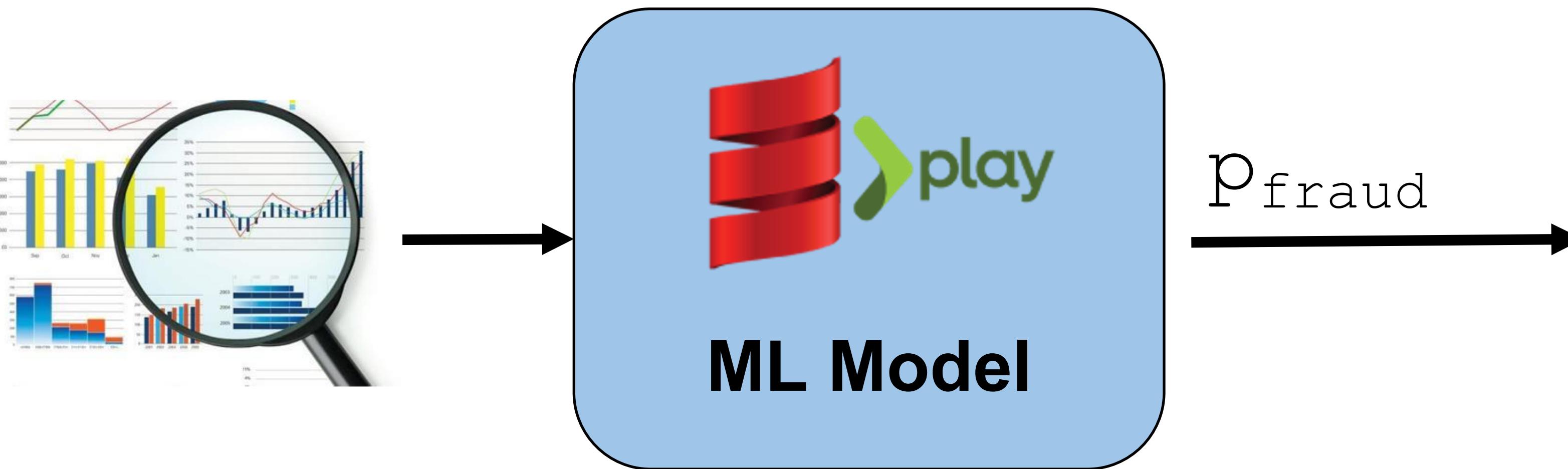
# INVESTIGATION



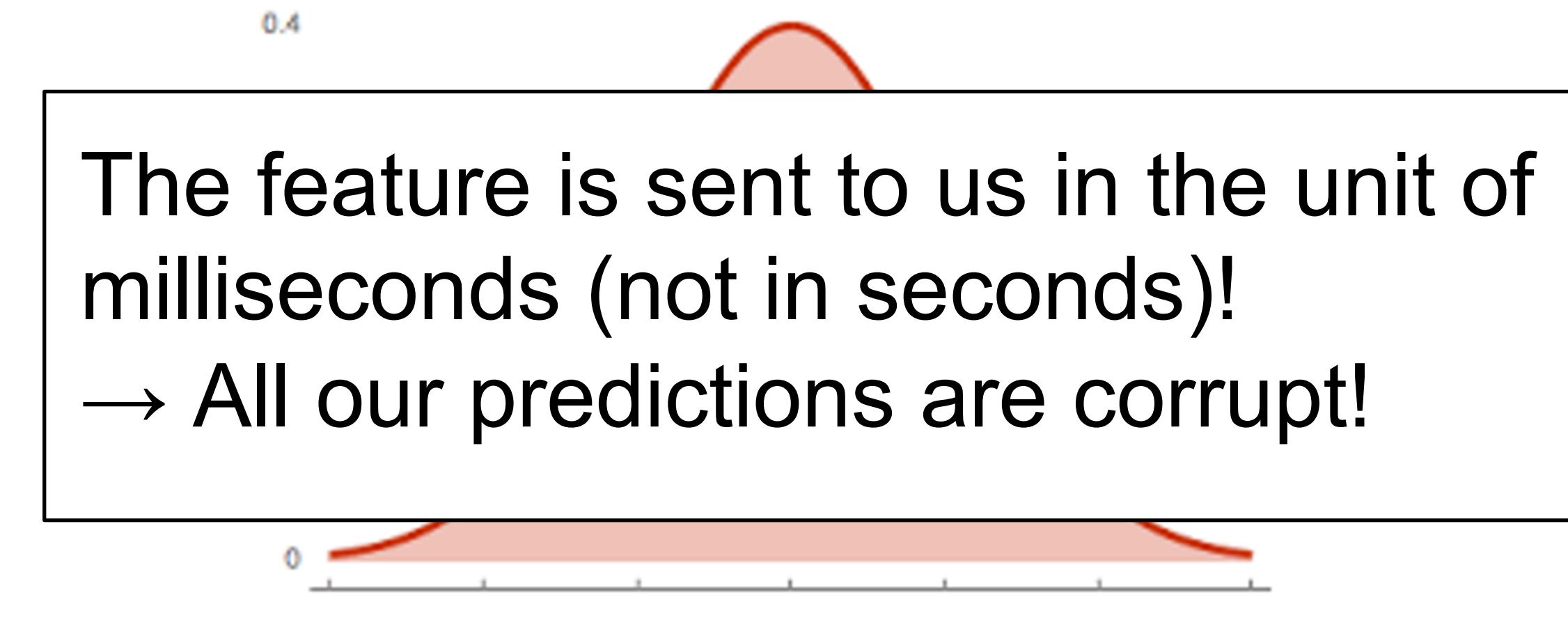
Time-to-order
300000
5000
120000
...



# INVESTIGATION



Time-to-order
300000
5000
120000
...



---

## PROBLEMS

1. We lost a lot of money.
2. We did not detect it in time.
3. We could have detected it in time and provided a fix.

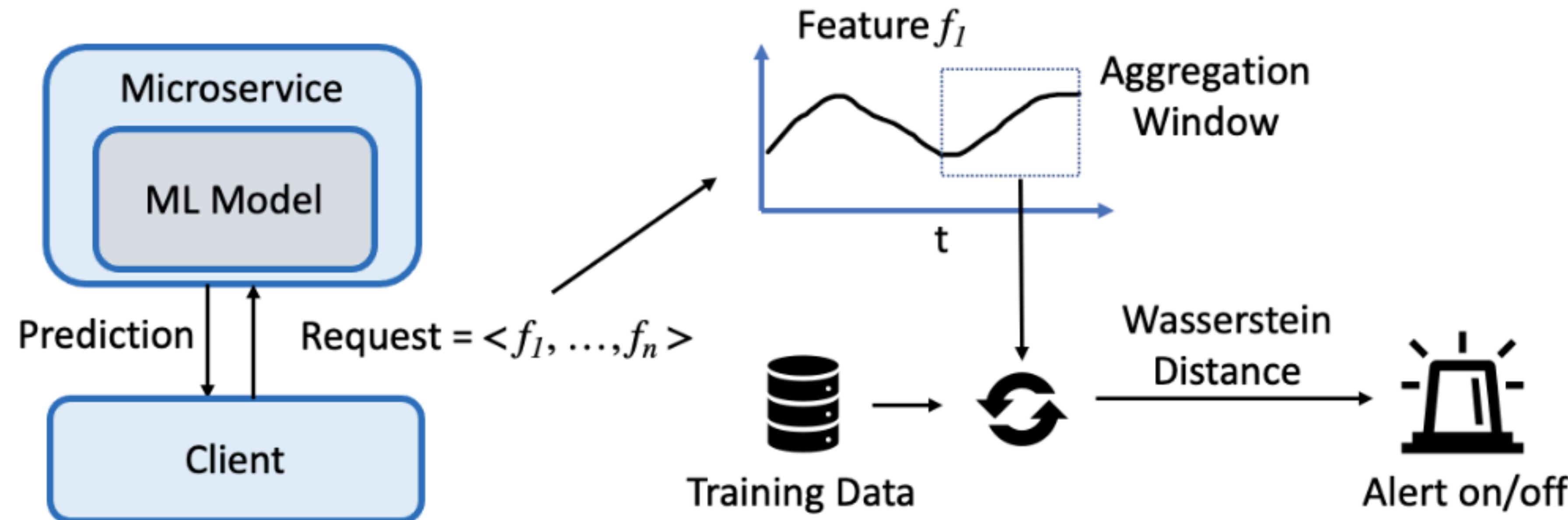
---

# CONCLUSIONS

We need to make sure that the distributions of  
input features are (always) the same  
as in training.

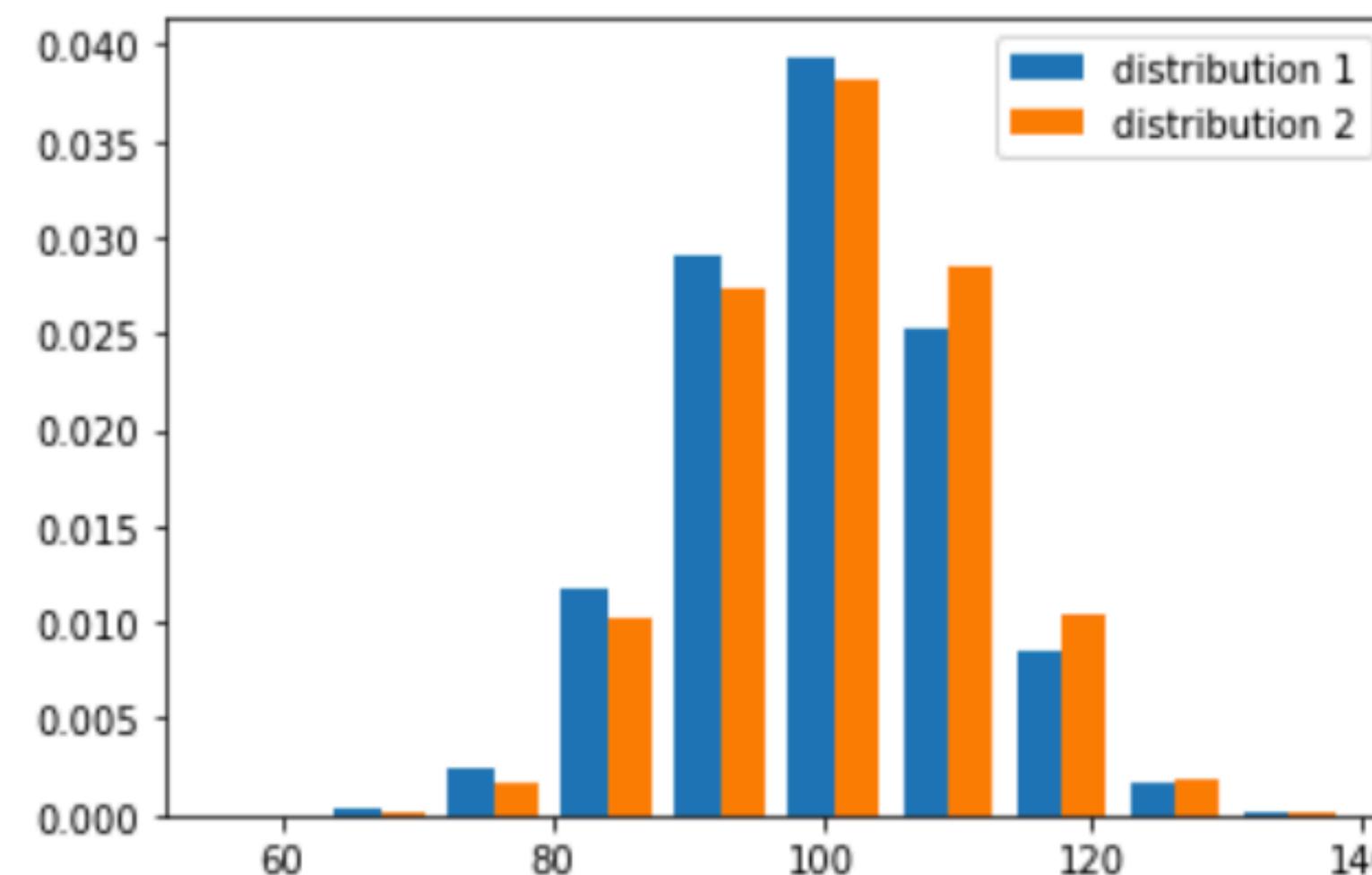
# Model Monitoring

- Wie setzt man ML-Monitoring um?
- Vergleiche jede  $t$  Sekunden die Verteilung der Features zwischen Test- und Livedaten:

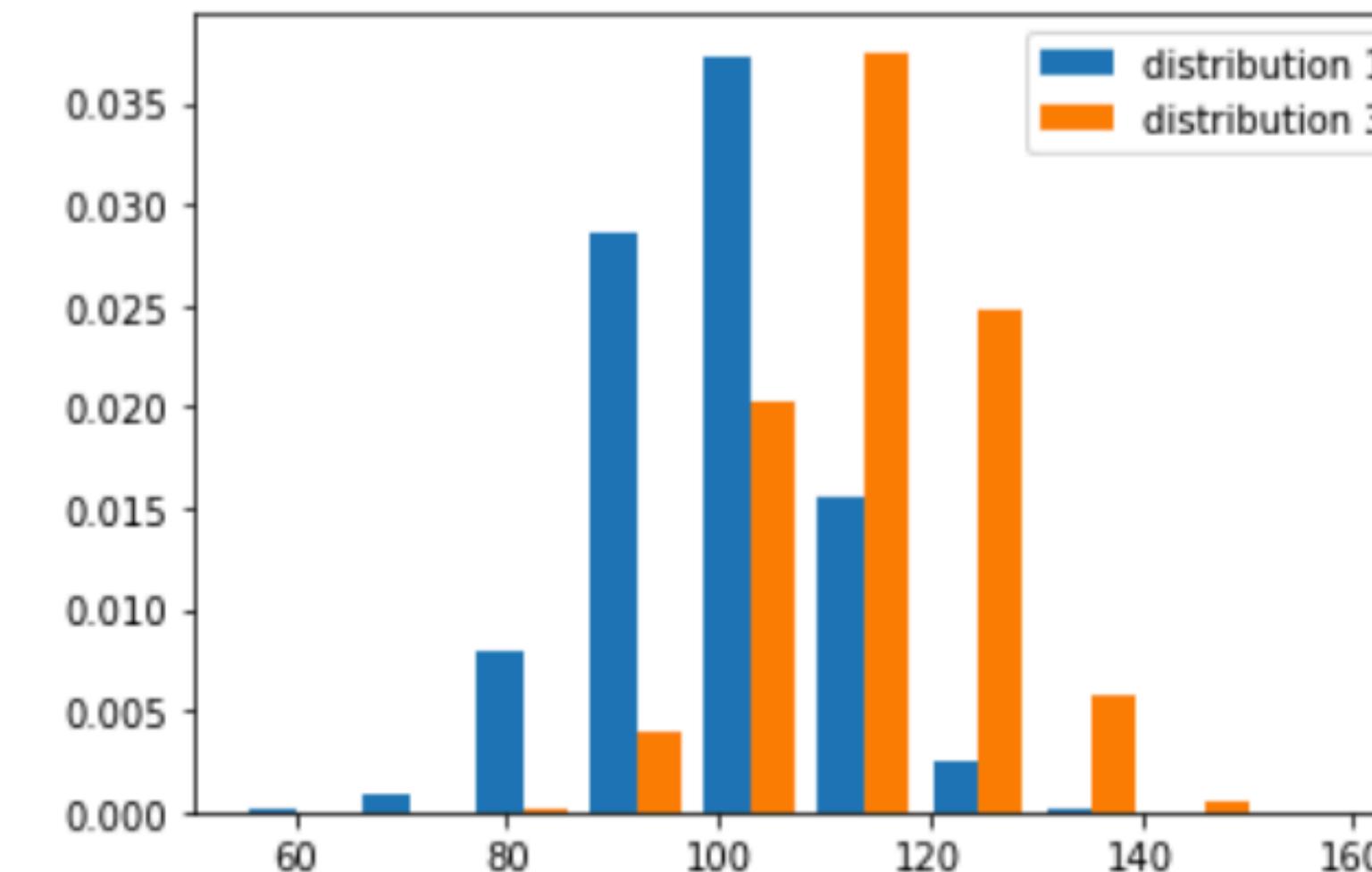


# Model Monitoring

- Wie vergleicht man die Verteilung der Features zwischen Test- und Livedaten?  
Zum Beispiel über Wasserstein-Distanz (WD):



(a) Similar distributions, WD=0.9



(b) Different distributions, WD=15

# Fragen?

# Let's stay in touch

Wenn Sie nach diesem Workshop ML-Methoden einsetzen wollen, gibt es mehrere Möglichkeiten bei denen ich unterstützen kann:

1. Studentische Abschlussarbeiten oder Praxissemester
2. KI-Wissenstransfer Hochschule zu Unternehmen
  - KI-Möglichkeiten im Unternehmen evaluieren
  - Wissenstransfer durch Workshops, z.B. „Kurzeinführung KI“, „ML für Softwareentwickler“, „KI für Entscheidet“
  - Hilfe bei der Implementierung von ML-Methoden
  - Kostenlose Erstberatung über Steinbeistransferzentrum