Forecasting Lifetime Value (LTV) using PySTAN

About me

- **2022 ~ Today: Data Science Lead in Marketing Science at Meta**
    - Apply causal inference techniques to extract insight to help advertisers in increasing the effectiveness of their marketing investments
    - Maintainer of GeoLift, an open-source techniques for measuring marketing effectiveness

- **2017 ~ 2022: Staff Data Scientist in Wildlife Studios**
    - Tech lead for the Lifetime Value team
    - Developed the Lifetime Value models for launching 5 games
    - Implemented ads monetization algorithm which increases daily ads revenue

Topics

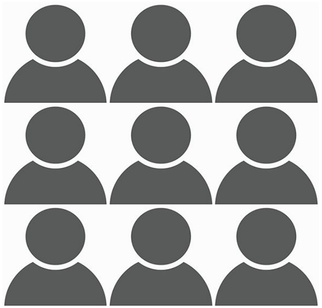
- Introduction to Advertisement
  - Modelling advertisement for digital products
  - Finding the optimal bid for your marketing campaigns
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - What is PySTAN
  - The data used: Lifetime Value dataset from Kaggle
  - Description of the model
  - Implementation of the model in PySTAN
  - Achieving the same model with PyMC
  - Comparison between PySTAN and PyMC

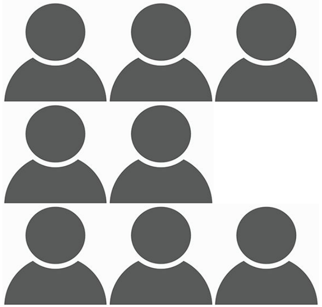
Topics

- Introduction to Advertisement
    - **Modelling advertisement for digital products**
    - Finding the optimal bid for your marketing campaigns
    - The role that uncertainty play in the marketing strategy

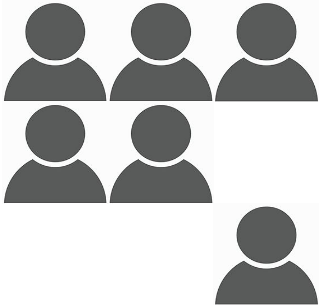
- Forecasting Lifetime Value with PySTAN
    - What is PySTAN
    - The data used: Lifetime Value dataset from Kaggle
    - Description of the model
    - Implementation of the model in PySTAN
    - Achieving the same model with PyMC
    - Comparison between PySTAN and PyMC

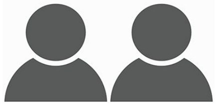


Day 1

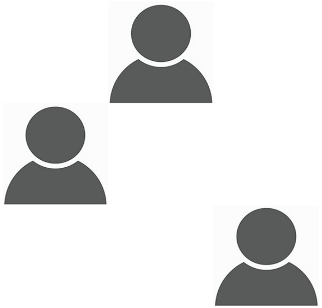


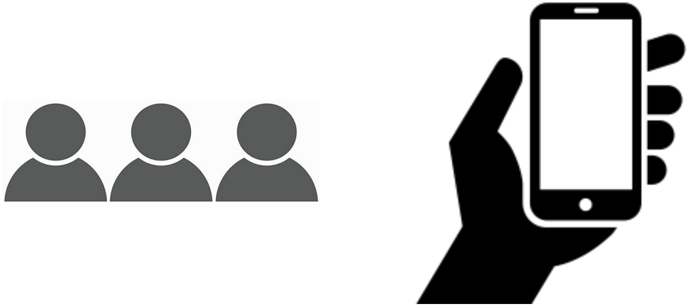
Day 2

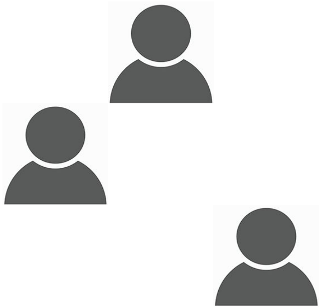


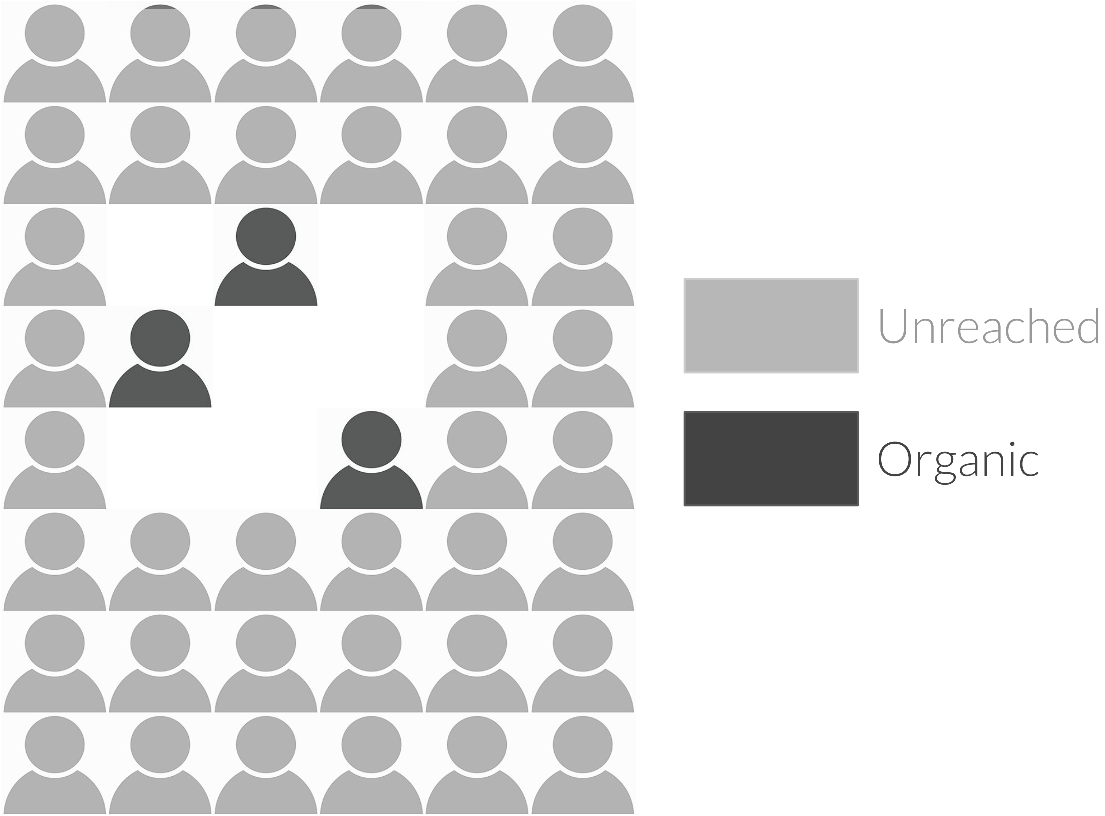


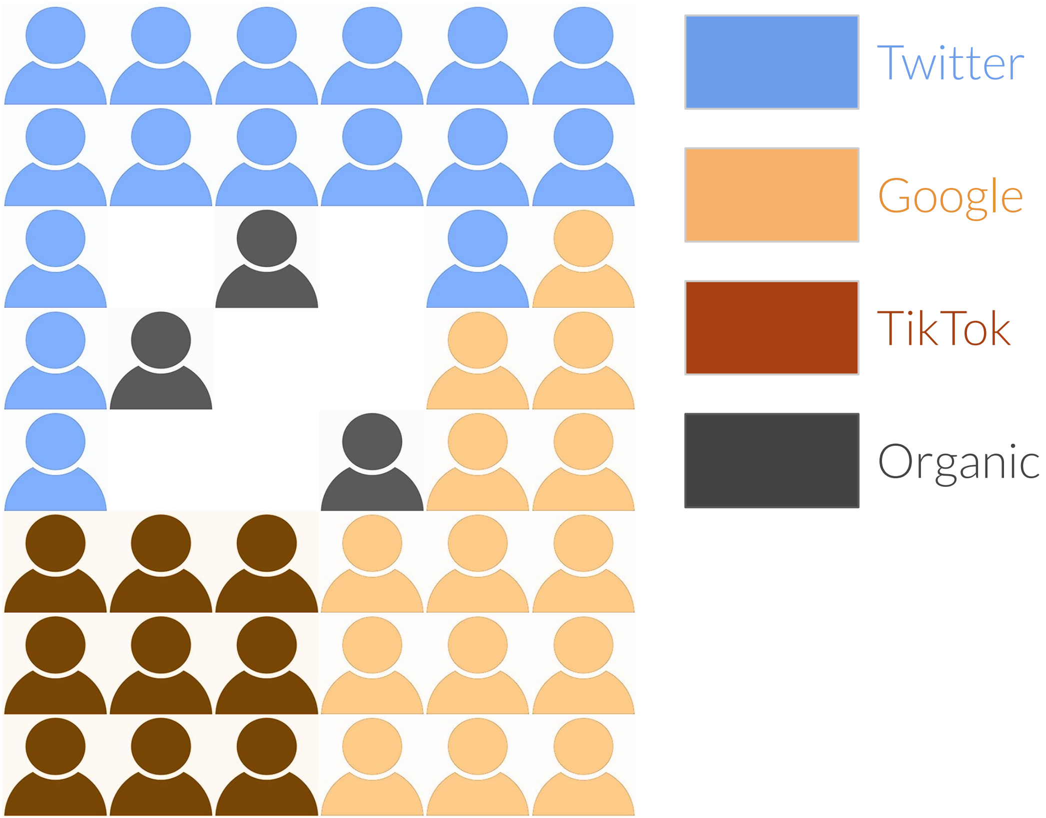
Day …

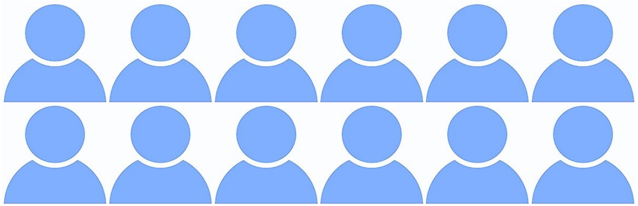


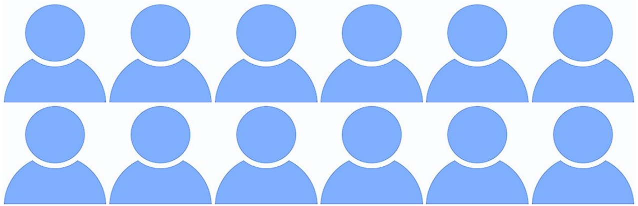


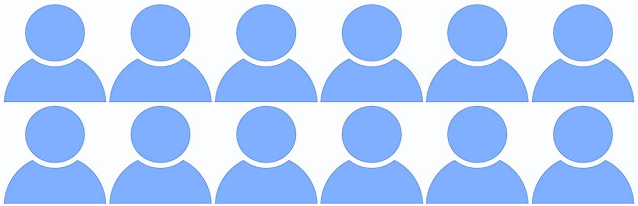


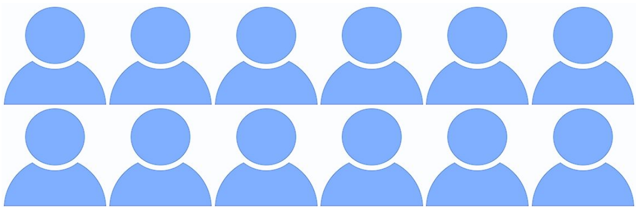

Unreached
Organic

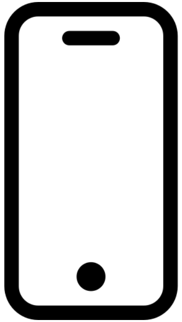

Twitter
Google
TikTok
Organic

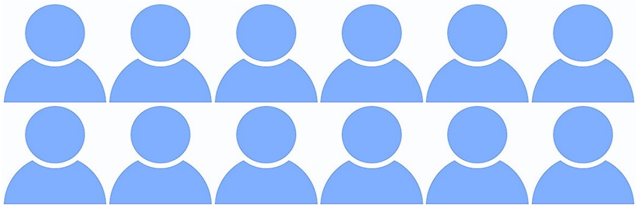


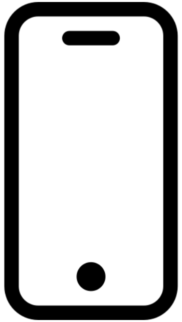


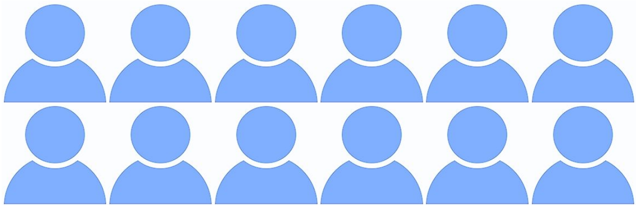


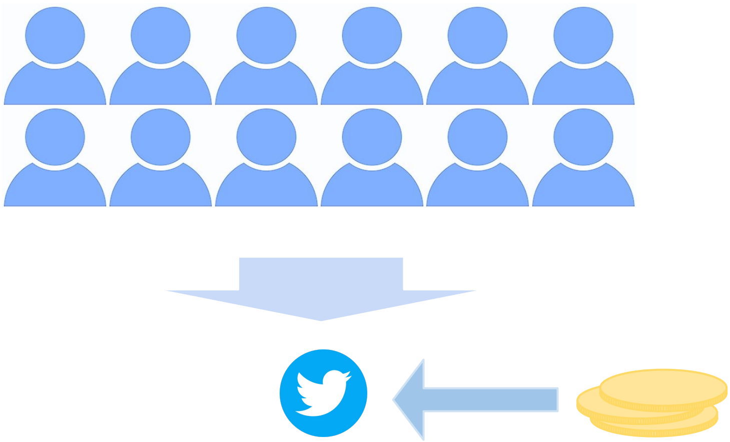


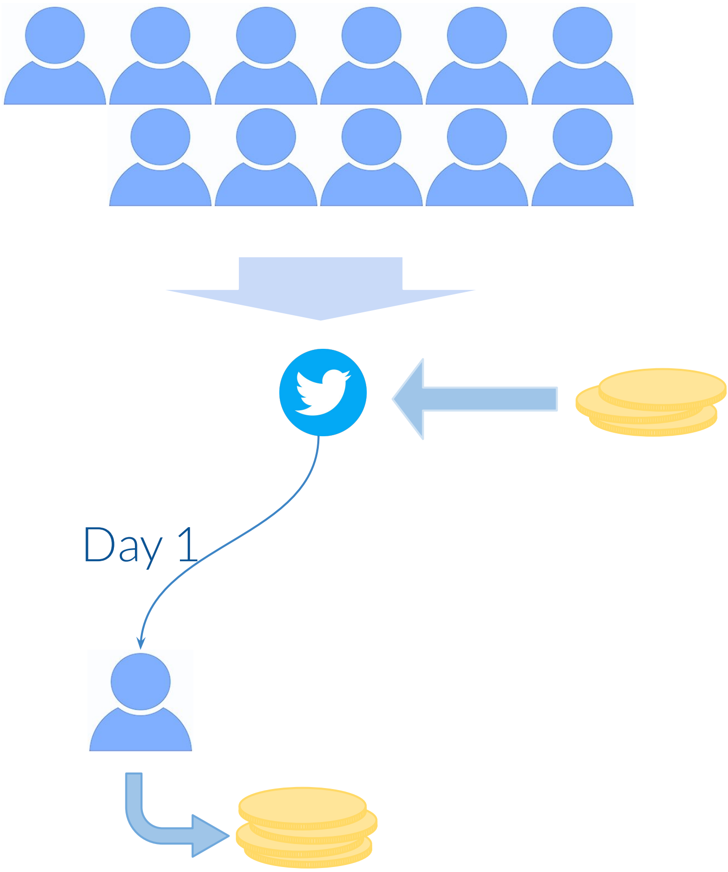


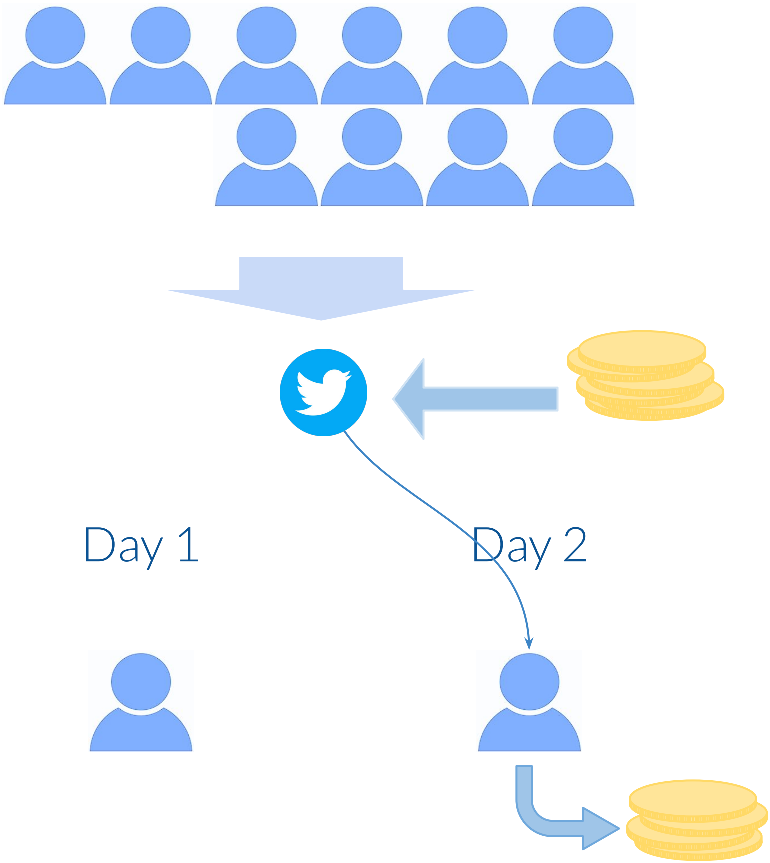


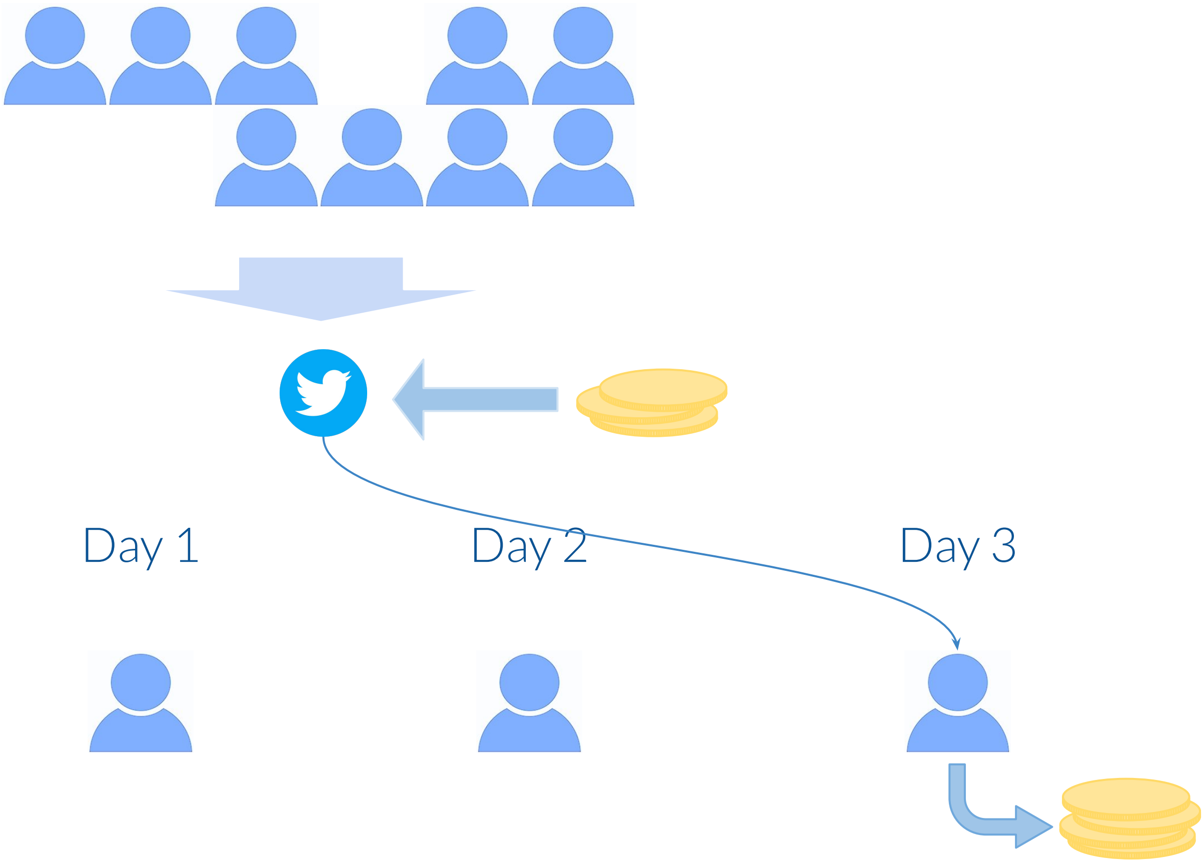


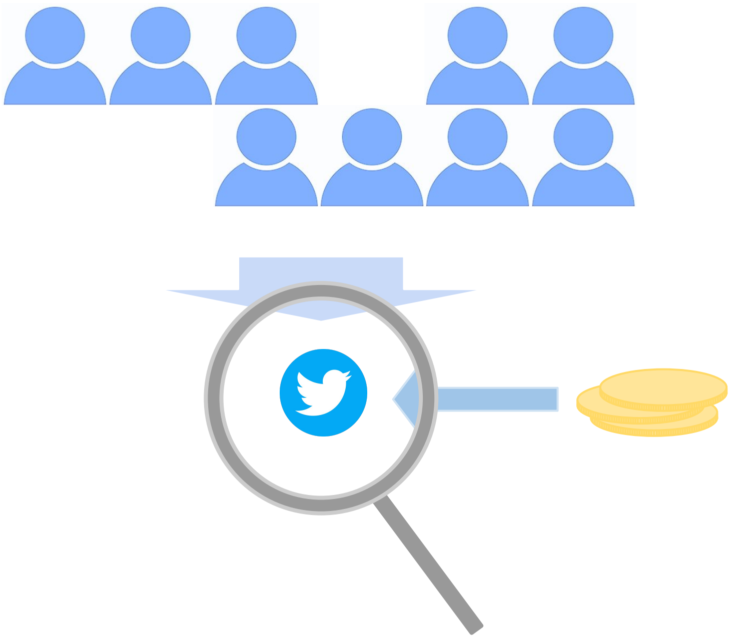


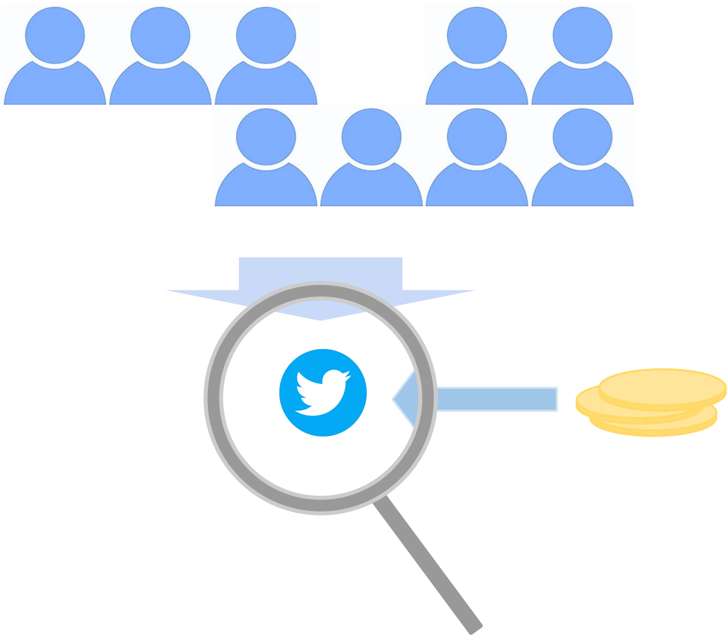


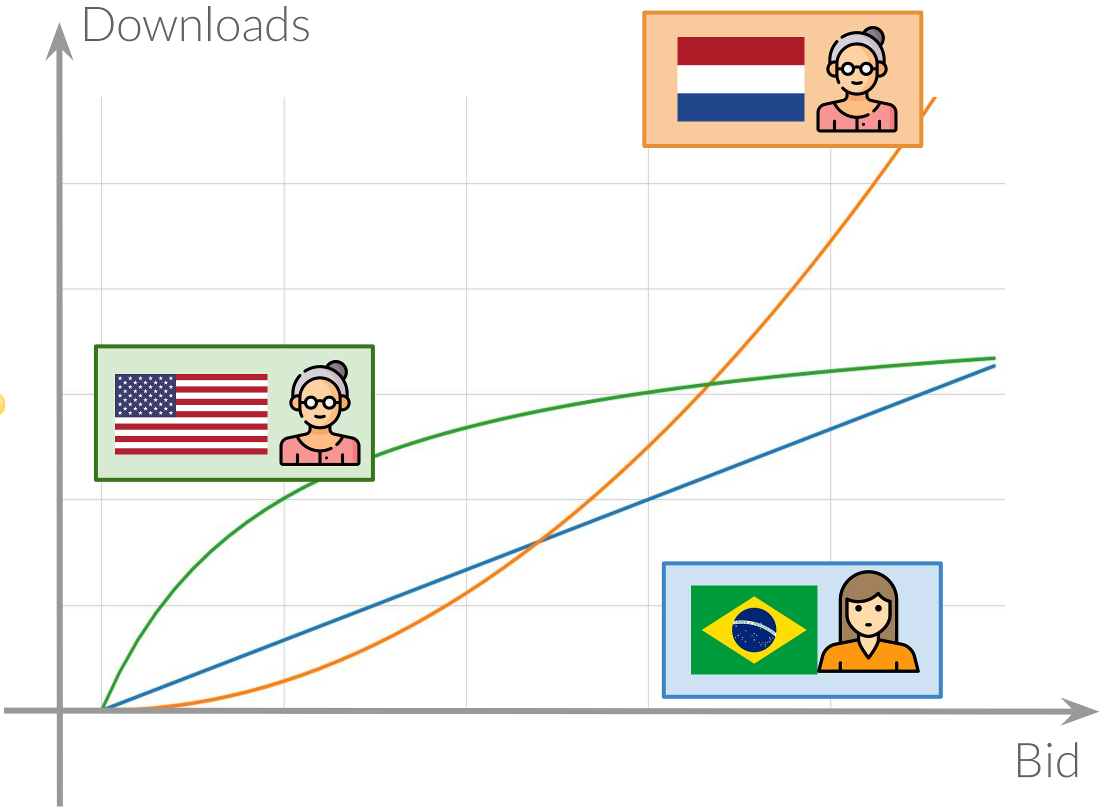

Day 1

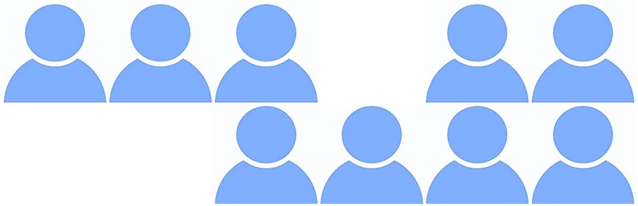

Day 1
Day 2

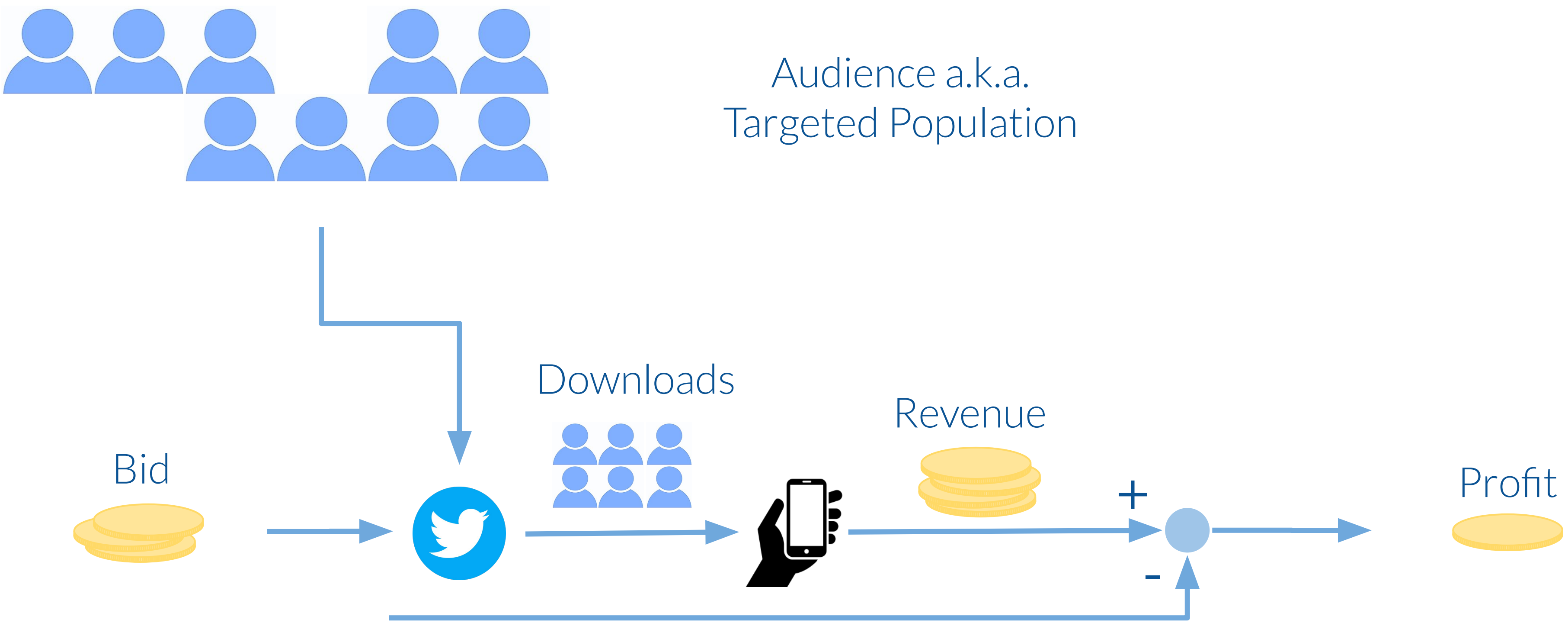

Day 1
Day 2
Day 3

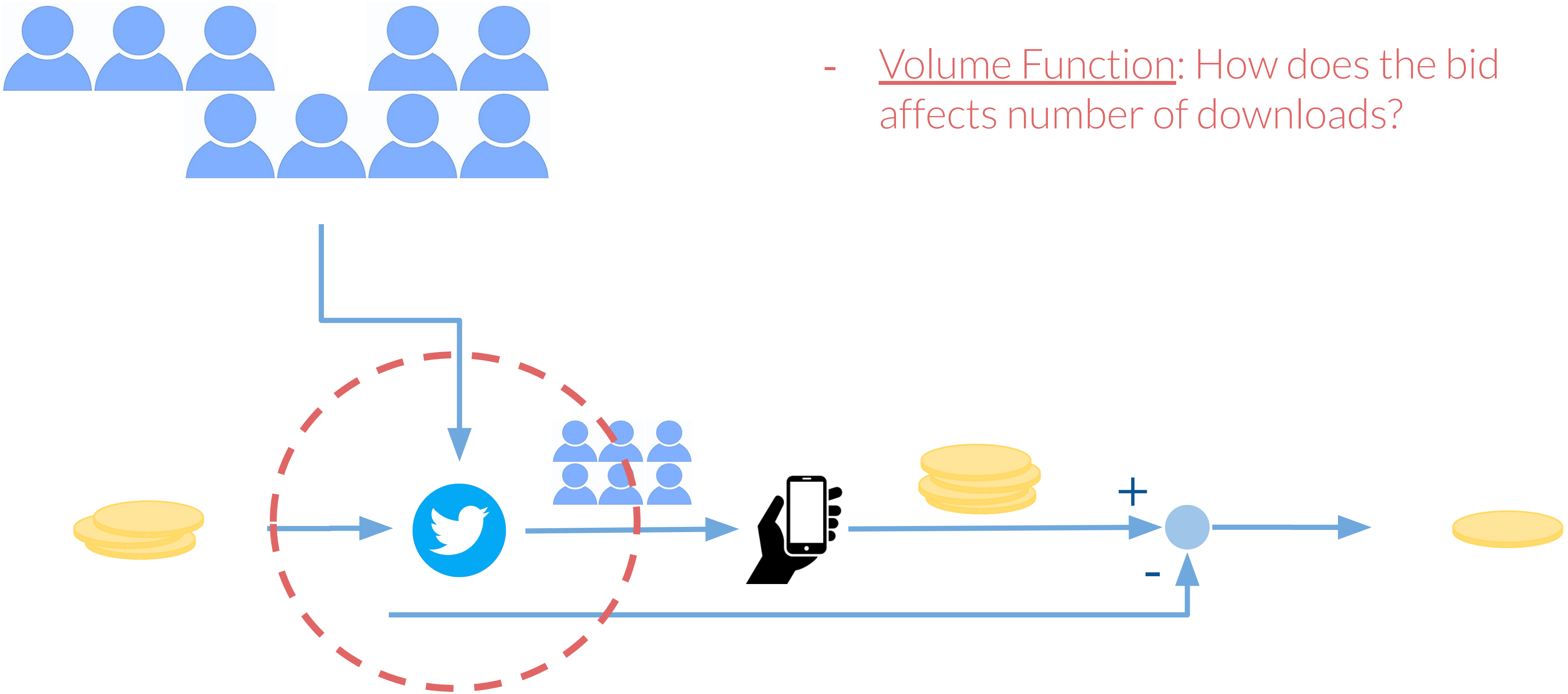




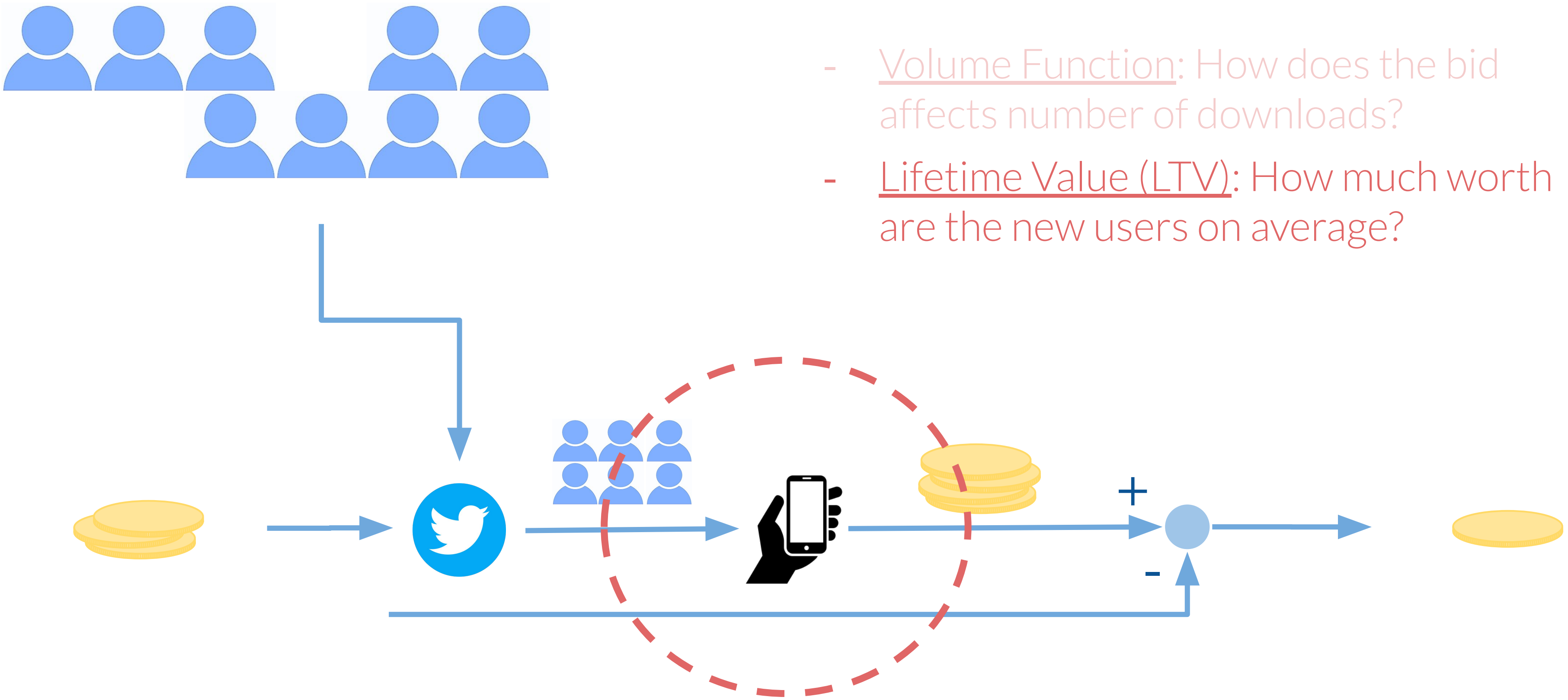

Downloads
Bid



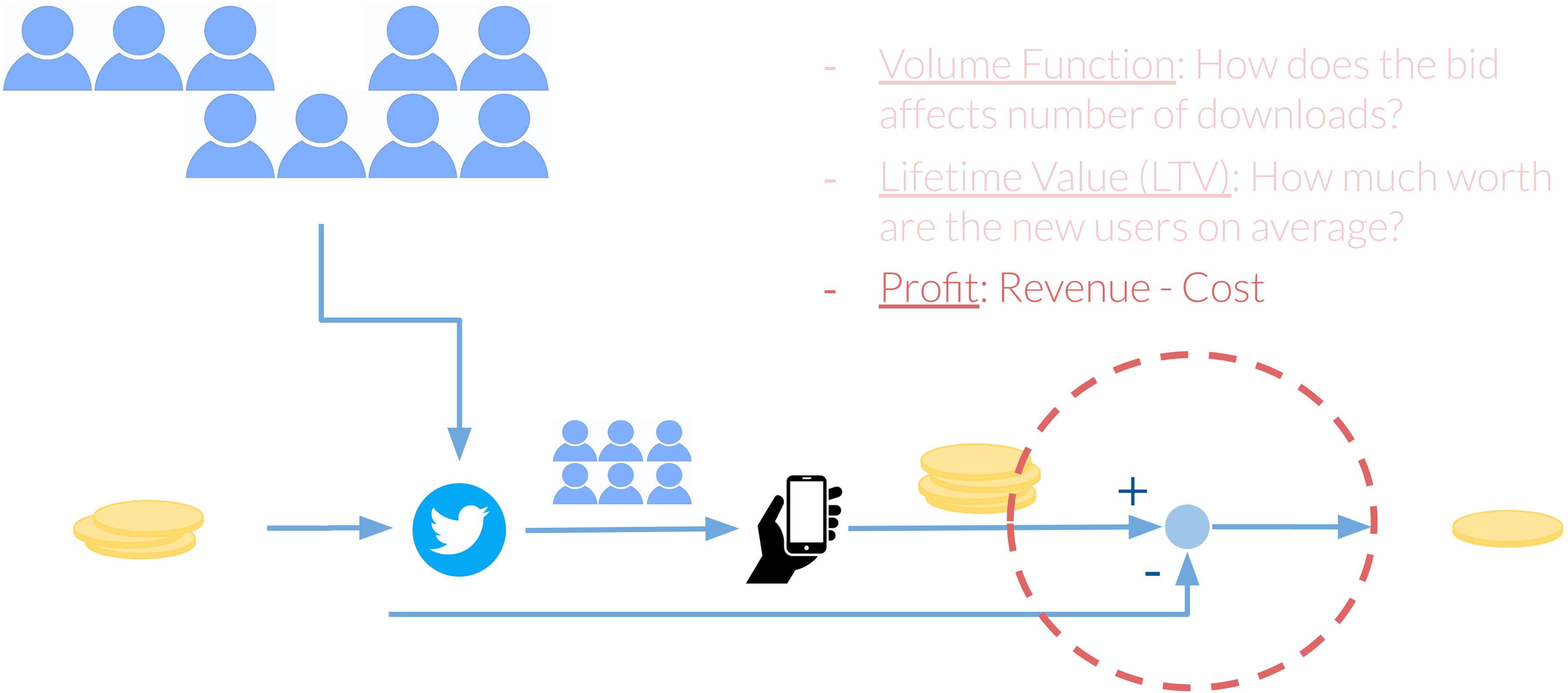

Audience a.k.a.
Targeted Population
Downloads
Revenue
Bid
+
-
Profit



- Volume Function: How does the bid affects number of downloads?

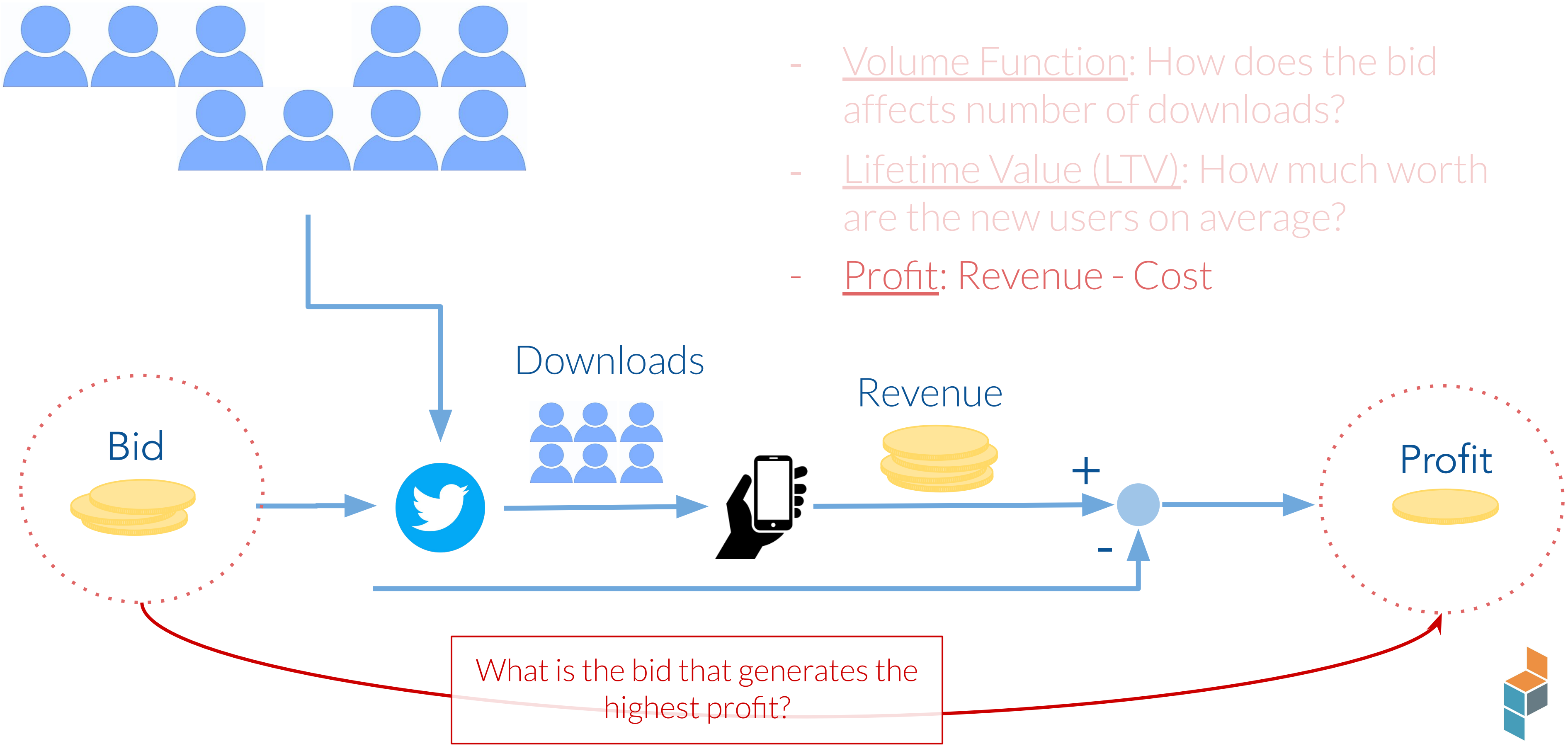


- Volume Function: How does the bid affects number of downloads?
- Lifetime Value (LTV): How much worth are the new users on average?



- Volume Function: How does the bid affects number of downloads?
- Lifetime Value (LTV): How much worth are the new users on average?
- Profit: Revenue - Cost

- Volume Function: How does the bid affects number of downloads?
- Lifetime Value (LTV): How much worth are the new users on average?
- Profit: Revenue - Cost

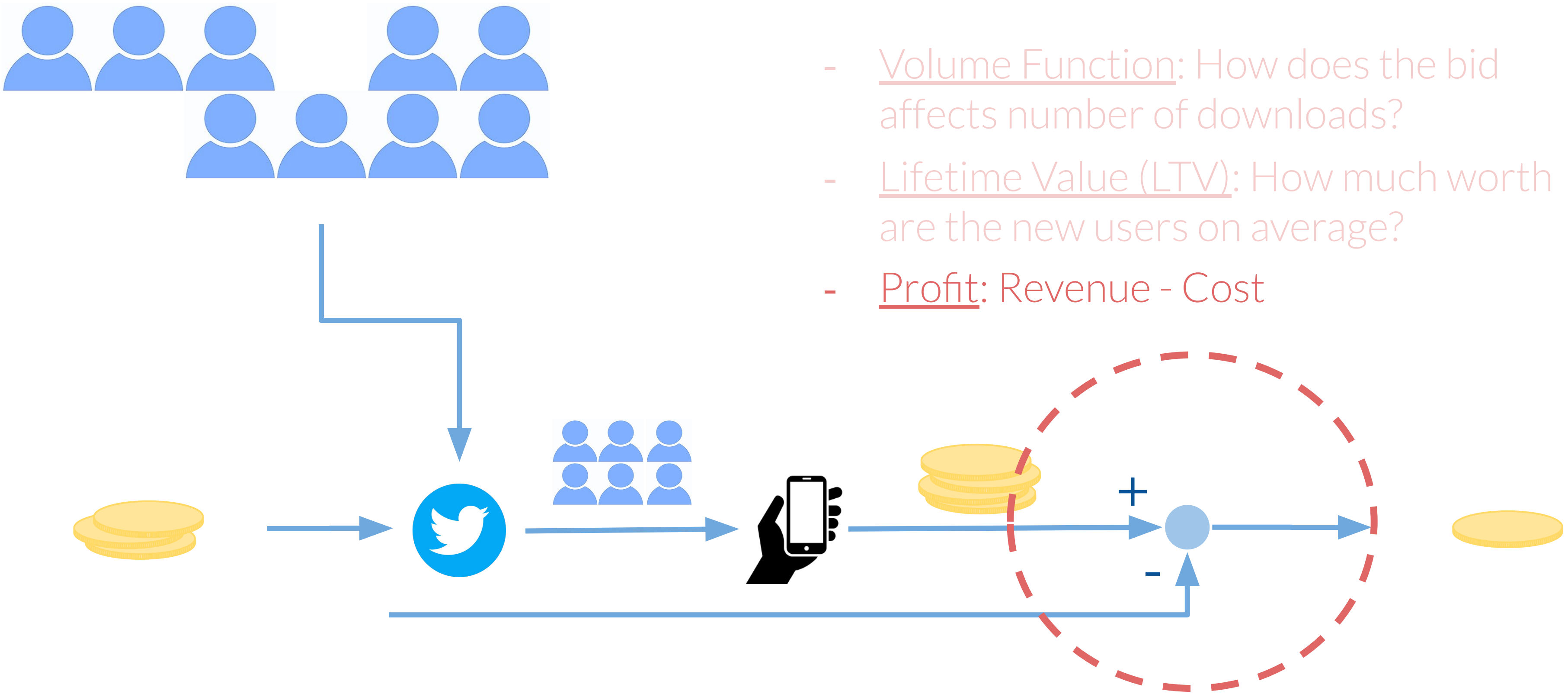


Topics

- Introduction to Advertisement
  - Modelling advertisement for digital products
  - **Finding the optimal bid for your marketing campaigns**
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - What is PySTAN
  - The data used: Lifetime Value dataset from Kaggle
  - Description of the model
  - Implementation of the model in PySTAN
  - Achieving the same model with PyMC
  - Comparison between PySTAN and PyMC

- Volume Function: How does the bid affects number of downloads?
- Lifetime Value (LTV): How much worth are the new users on average?
- Profit: Revenue - Cost



$$Profit = Revenue - Cost$$

$$Profit = Revenue - Cost$$

$$Profit = LTV * Downloads - Bid * Downloads$$

$$Profit = Revenue - Cost$$

$$Profit = LTV * Downloads - Bid * Downloads$$

$$Profit = (LTV - Bid) * Downloads$$

$$Profit = Revenue - Cost$$

$$Profit = LTV * Downloads - Bid * Downloads$$

$$Profit = (LTV - Bid) * Downloads$$

$$Profit = (LTV - Bid) * Vol\,(Bid)$$

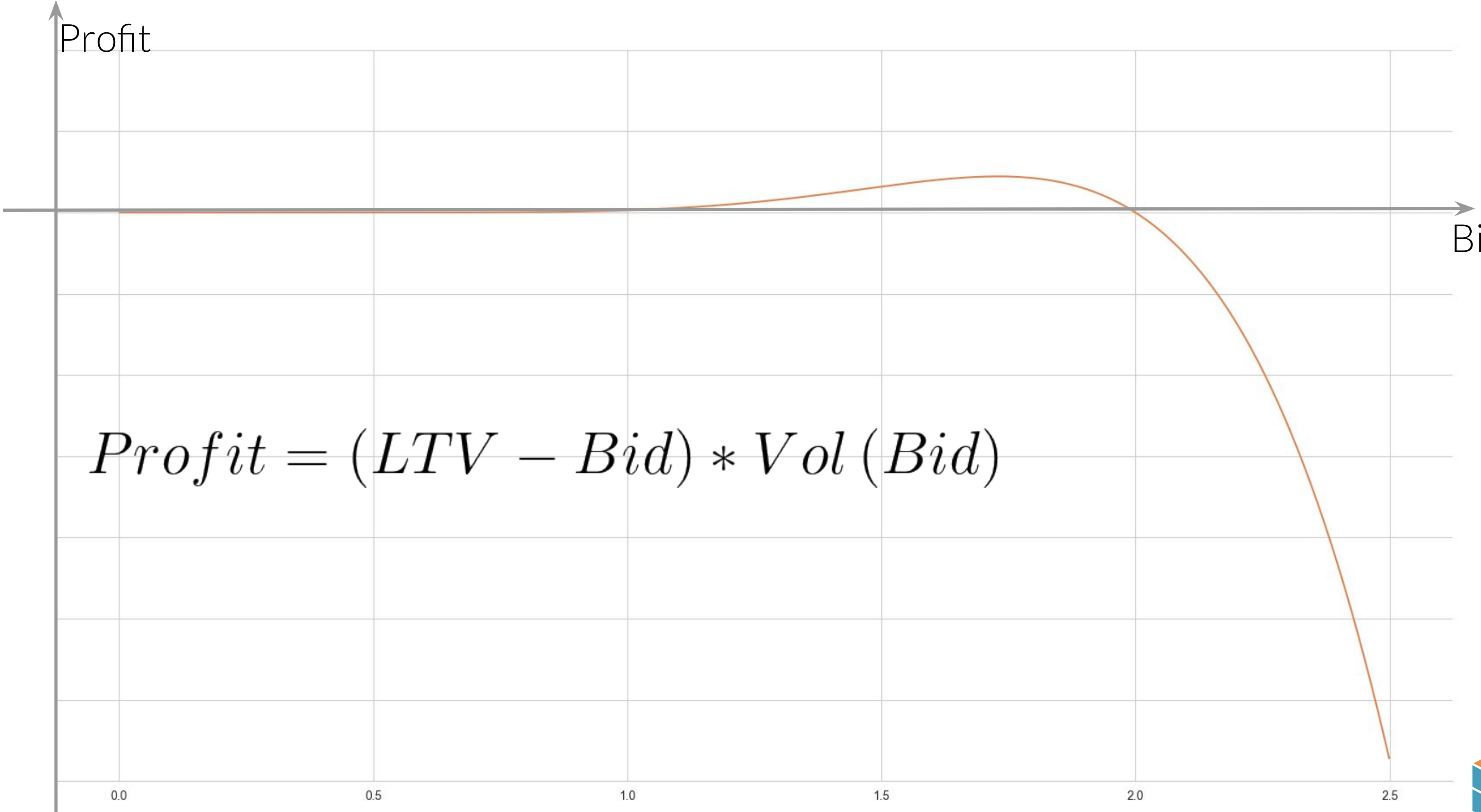
$$Profit = Revenue - Cost$$

$$Profit = LTV * Downloads - Bid * Downloads$$

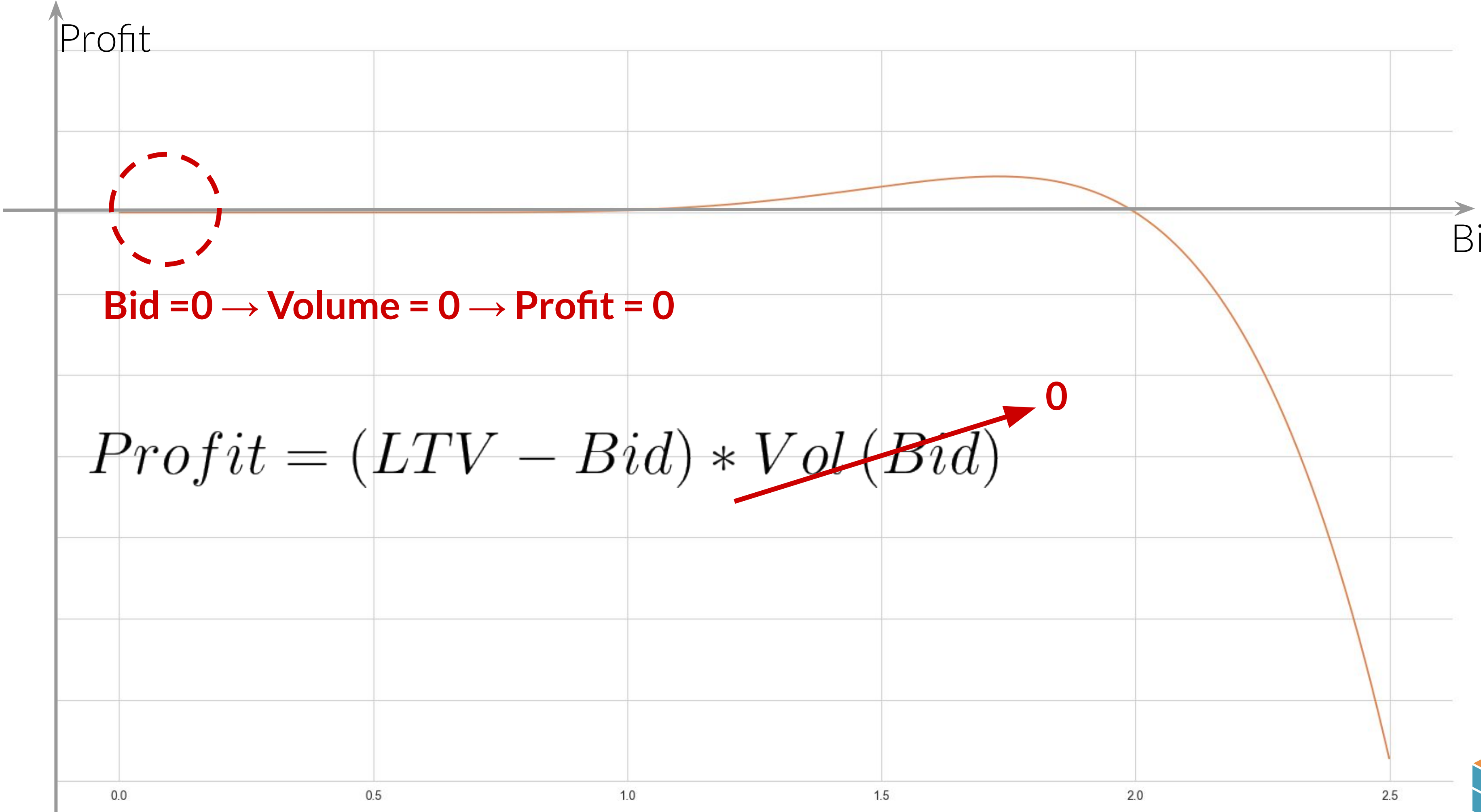
$$Profit = (LTV - Bid) * Downloads$$

$$Profit = (LTV - Bid) * Vol\,(Bid)$$

$$Profit = f(Bid)$$

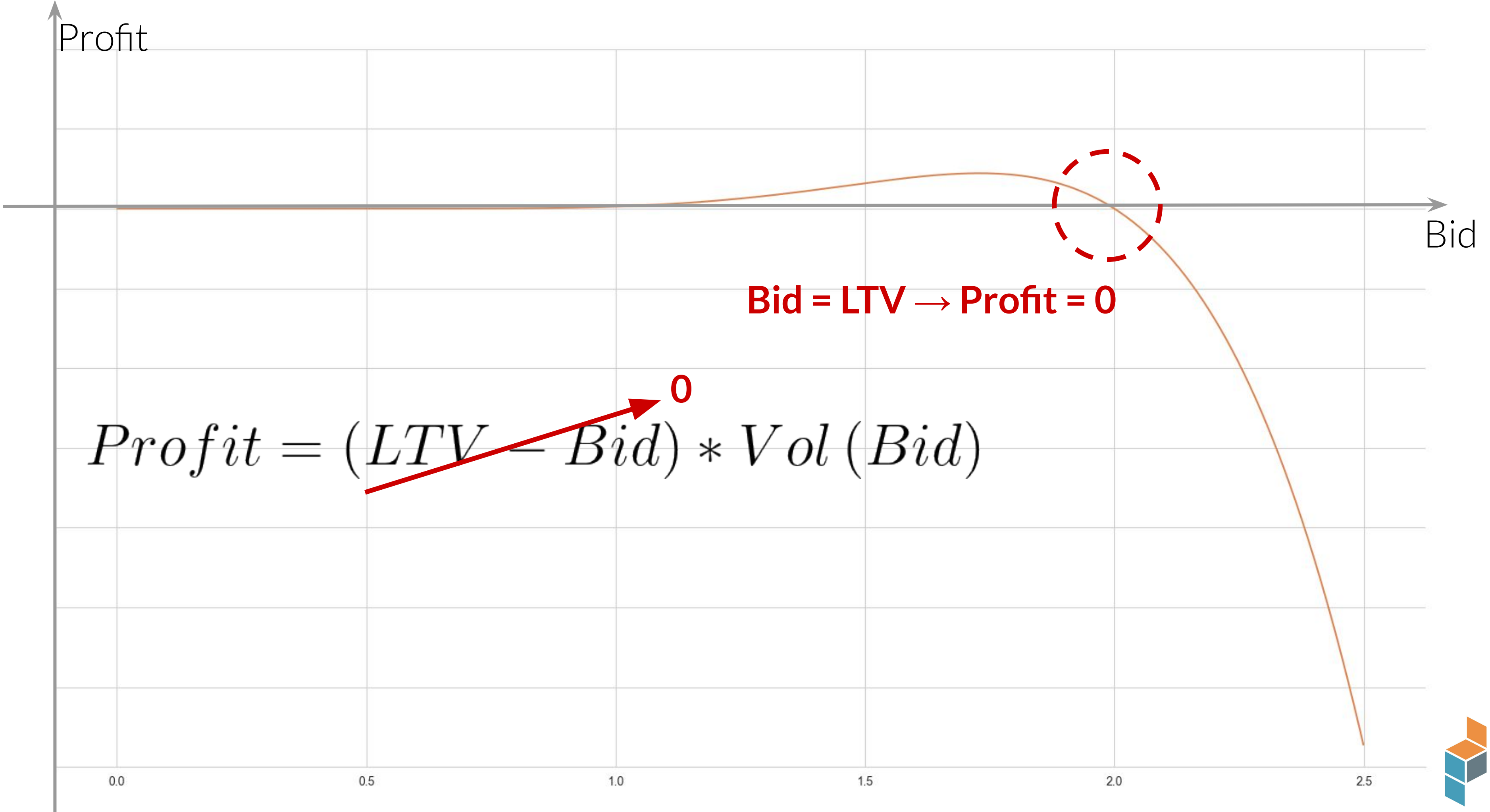


$$Profit = (LTV - Bid) * Vol\,(Bid)$$

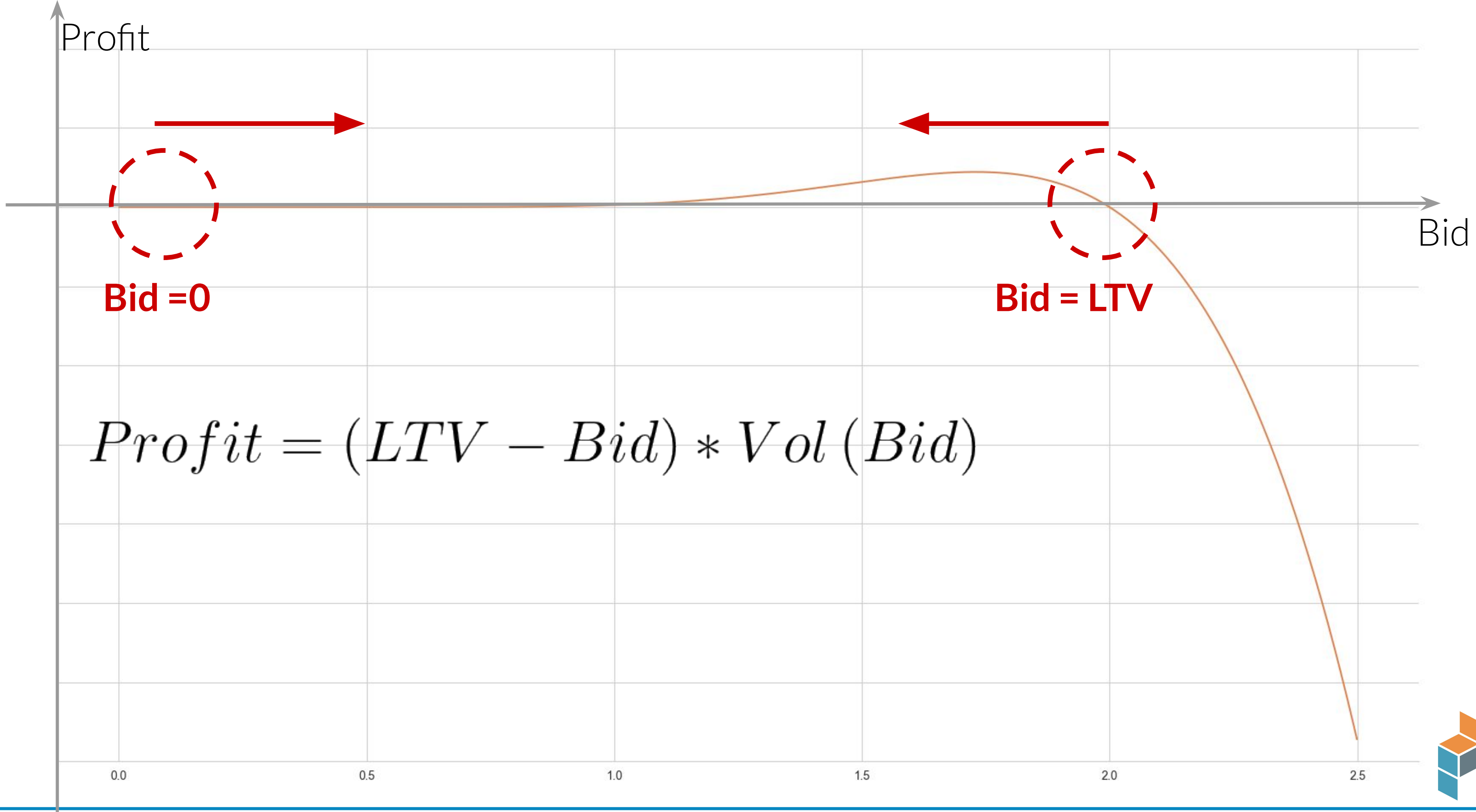


Bid =0 → Volume = 0 → Profit = 0

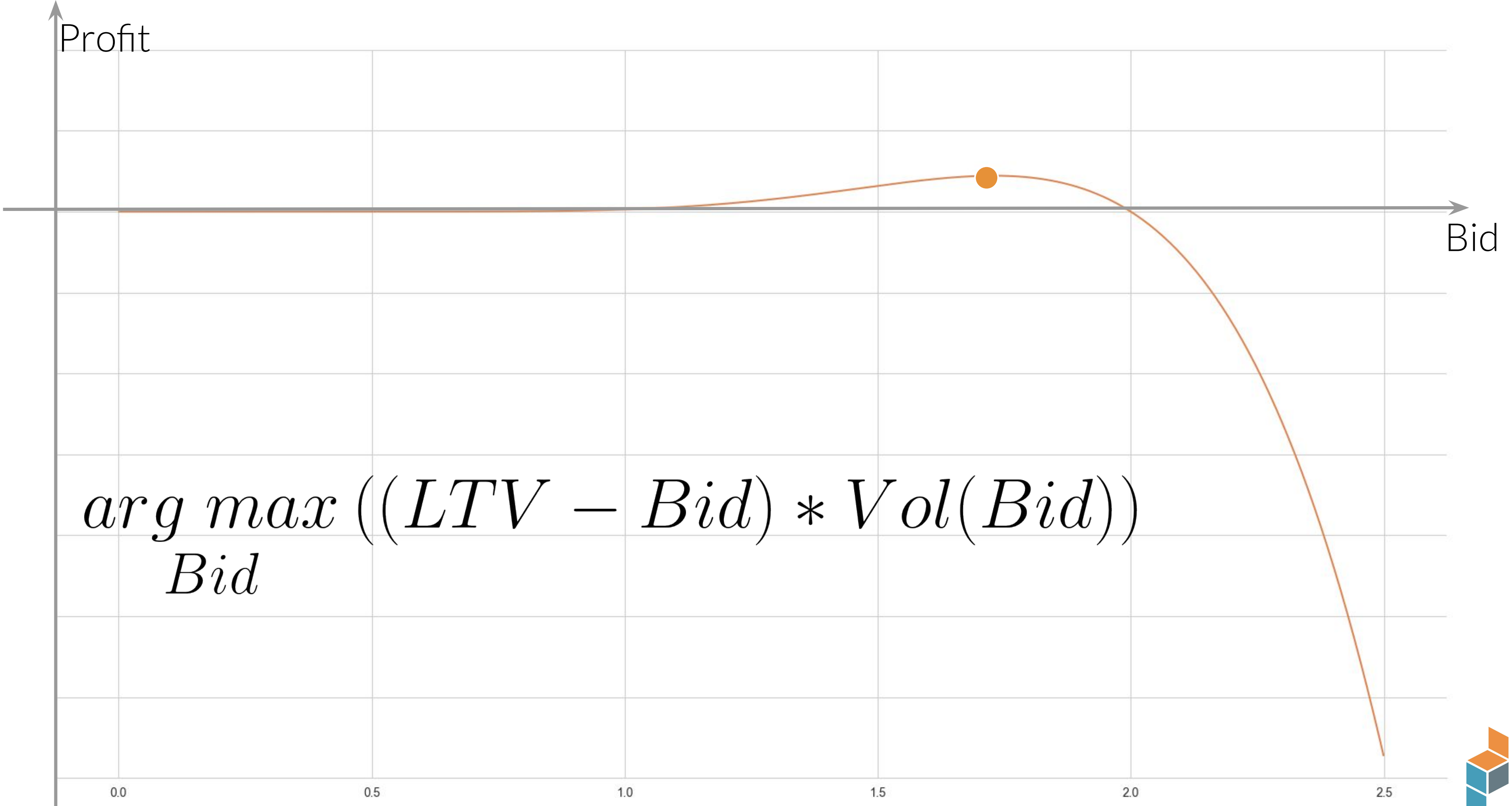
$$Profit = (LTV - Bid) * \cancelto{0}{Vol(Bid)}$$



$$Profit = (LTV - Bid) * Vol\,(Bid)$$



$$Profit = (LTV - Bid) * Vol\,(Bid)$$

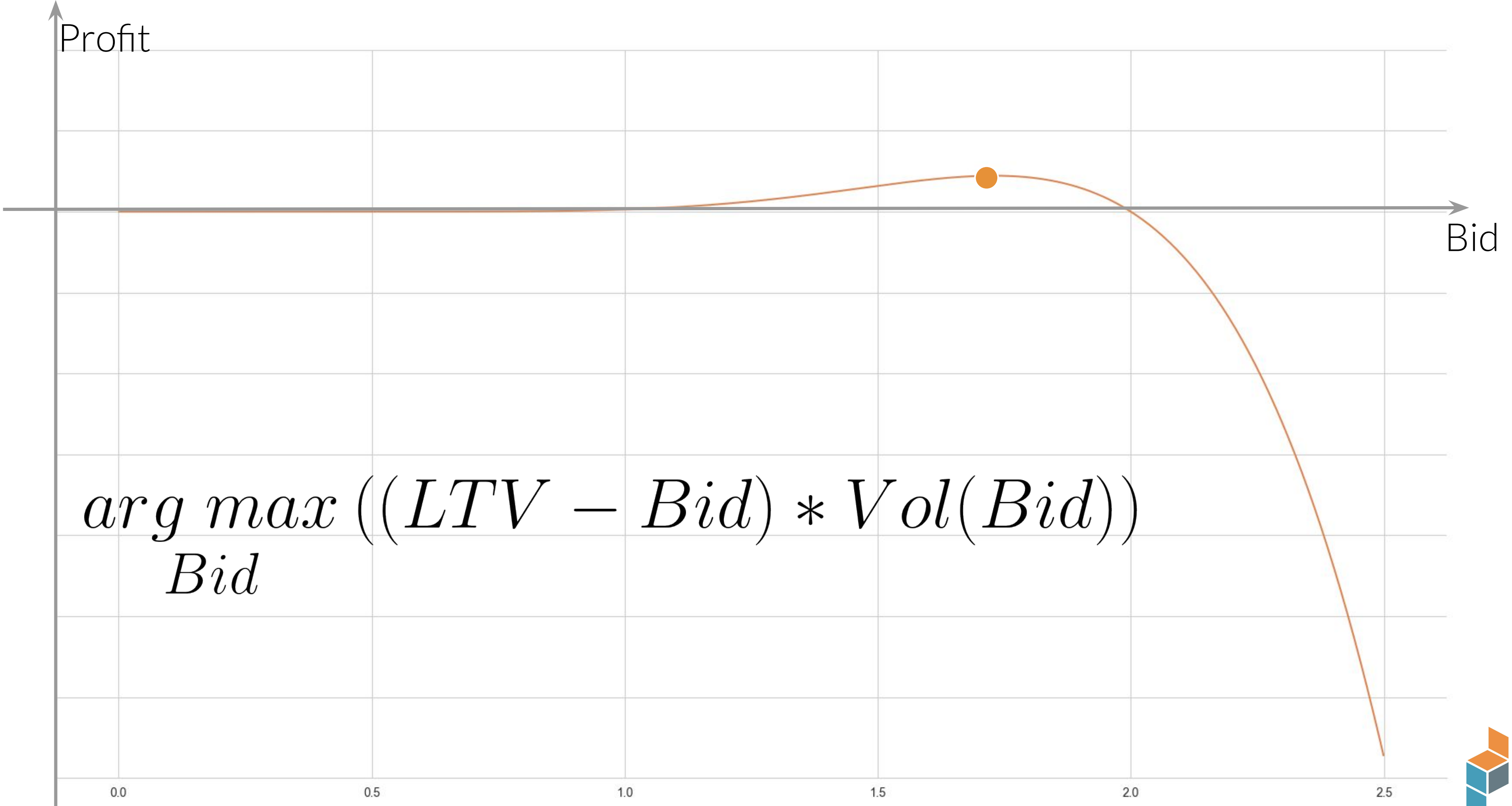


$$\underset{Bid}{arg\,max}\,((LTV - Bid) * Vol(Bid))$$

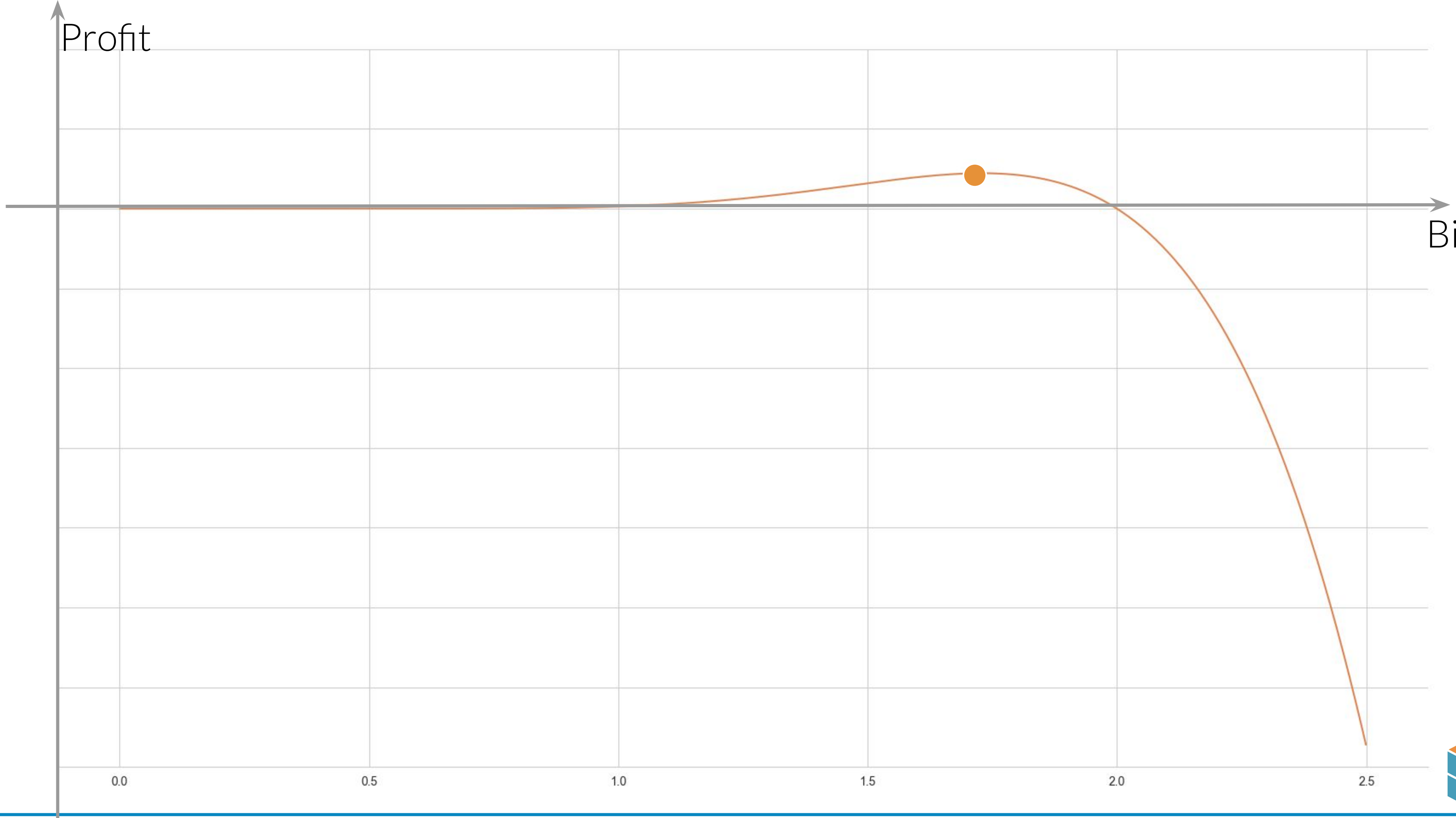
Topics

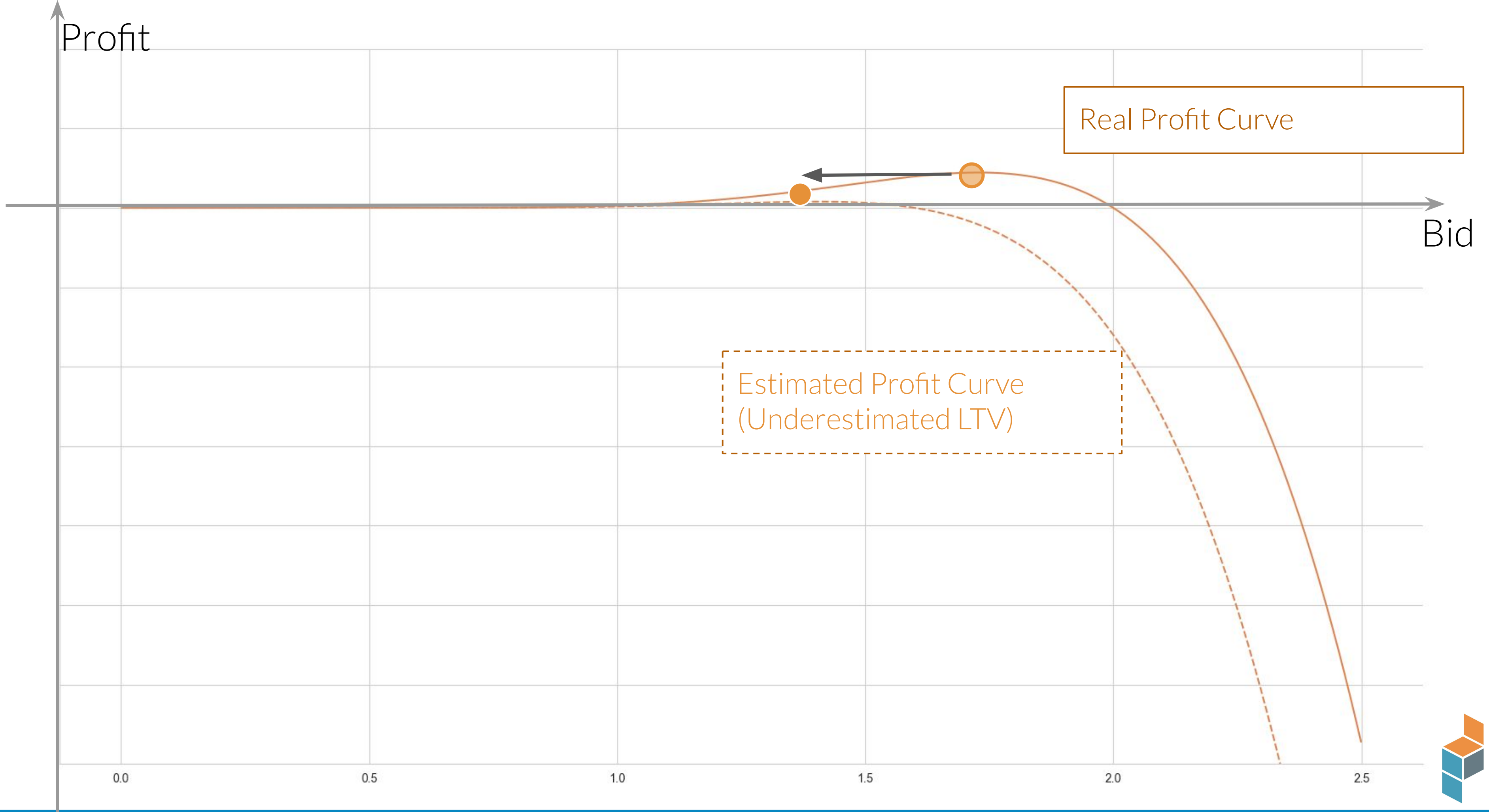
- Introduction to Advertisement
    - Modelling advertisement for digital products
    - Finding the optimal bid for your marketing campaigns
    - **The role that uncertainty play in the marketing strategy**

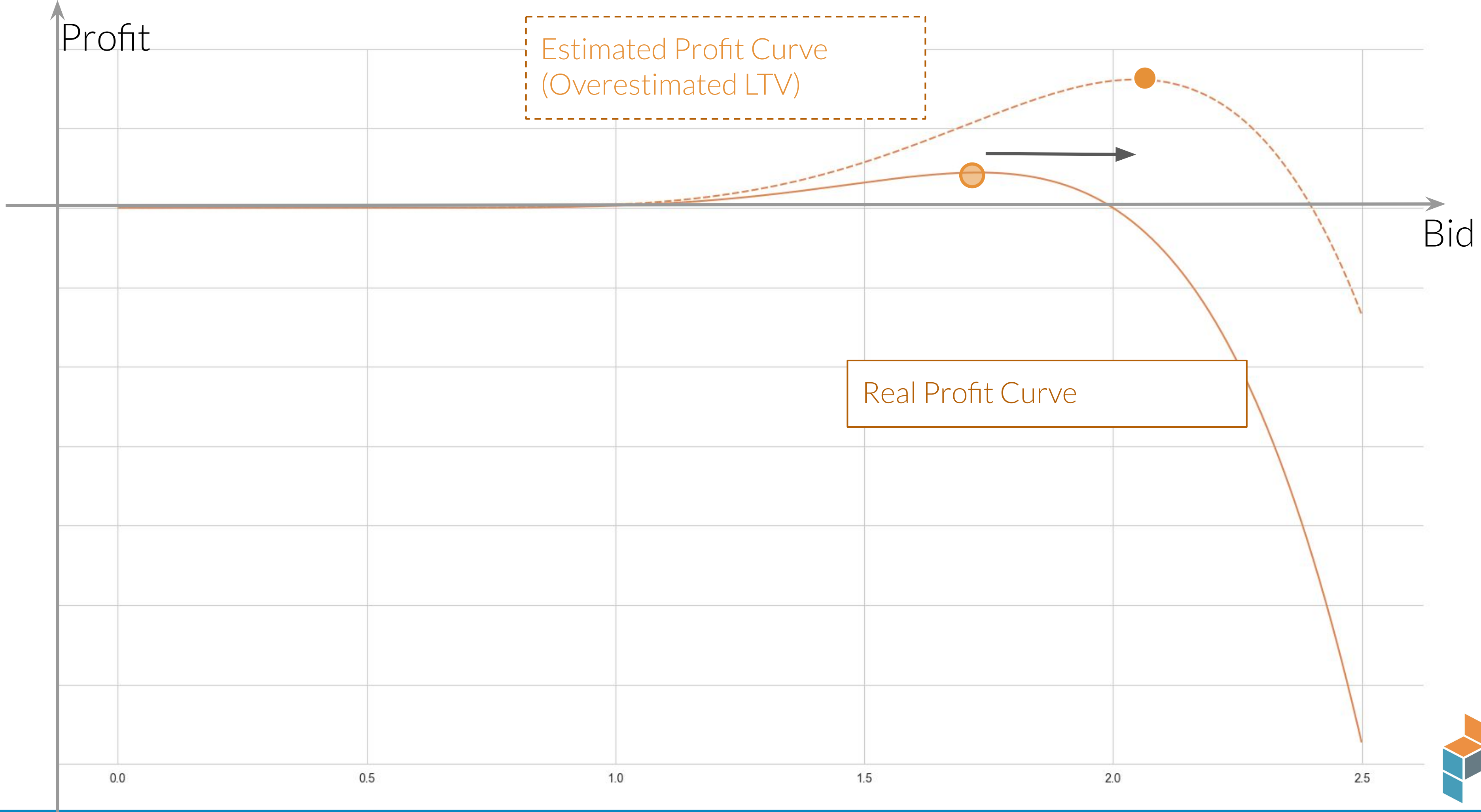
- Forecasting Lifetime Value with PySTAN
    - What is PySTAN
    - The data used: Lifetime Value dataset from Kaggle
    - Description of the model
    - Implementation of the model in PySTAN
    - Achieving the same model with PyMC
    - Comparison between PySTAN and PyMC

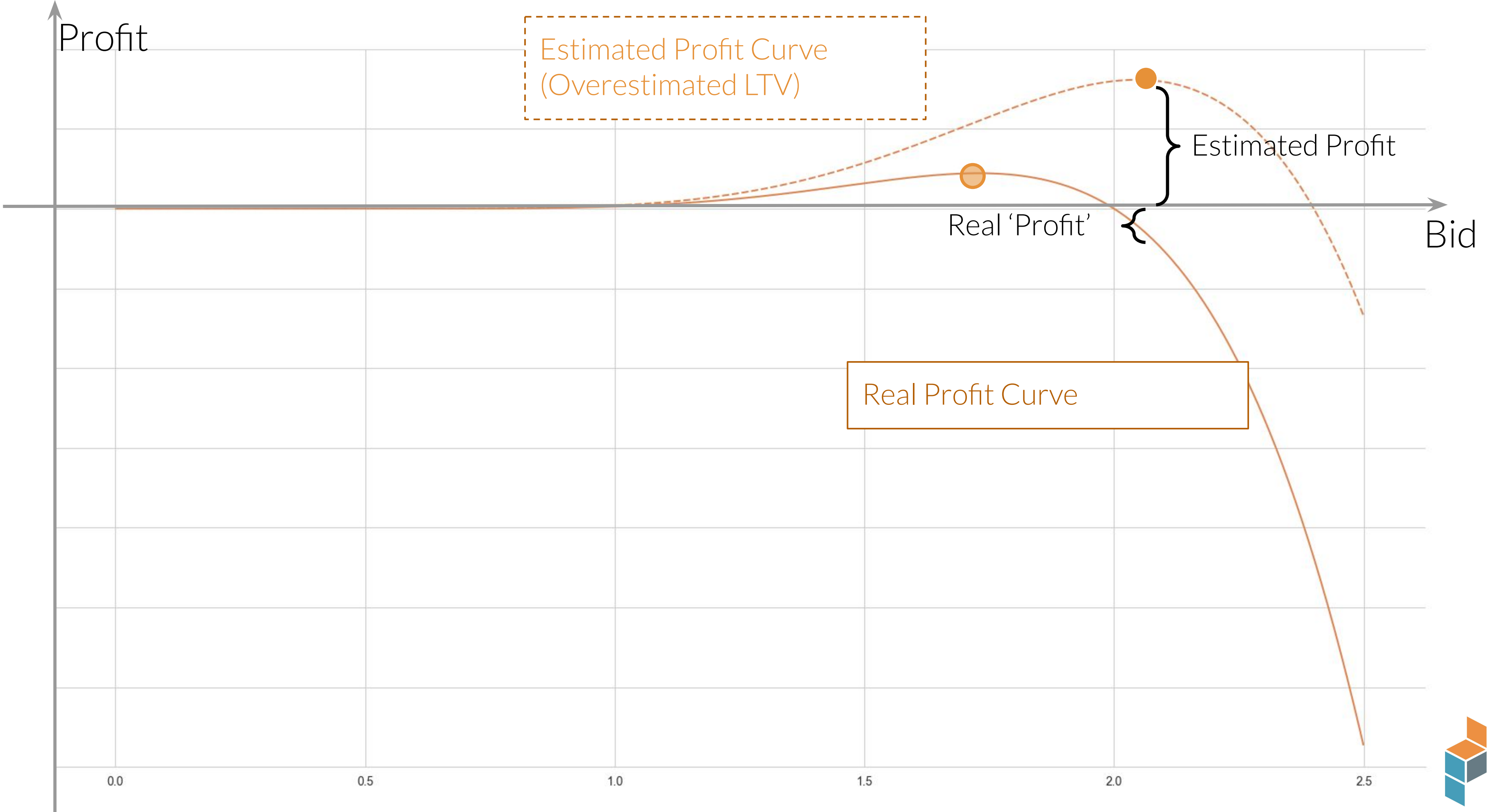


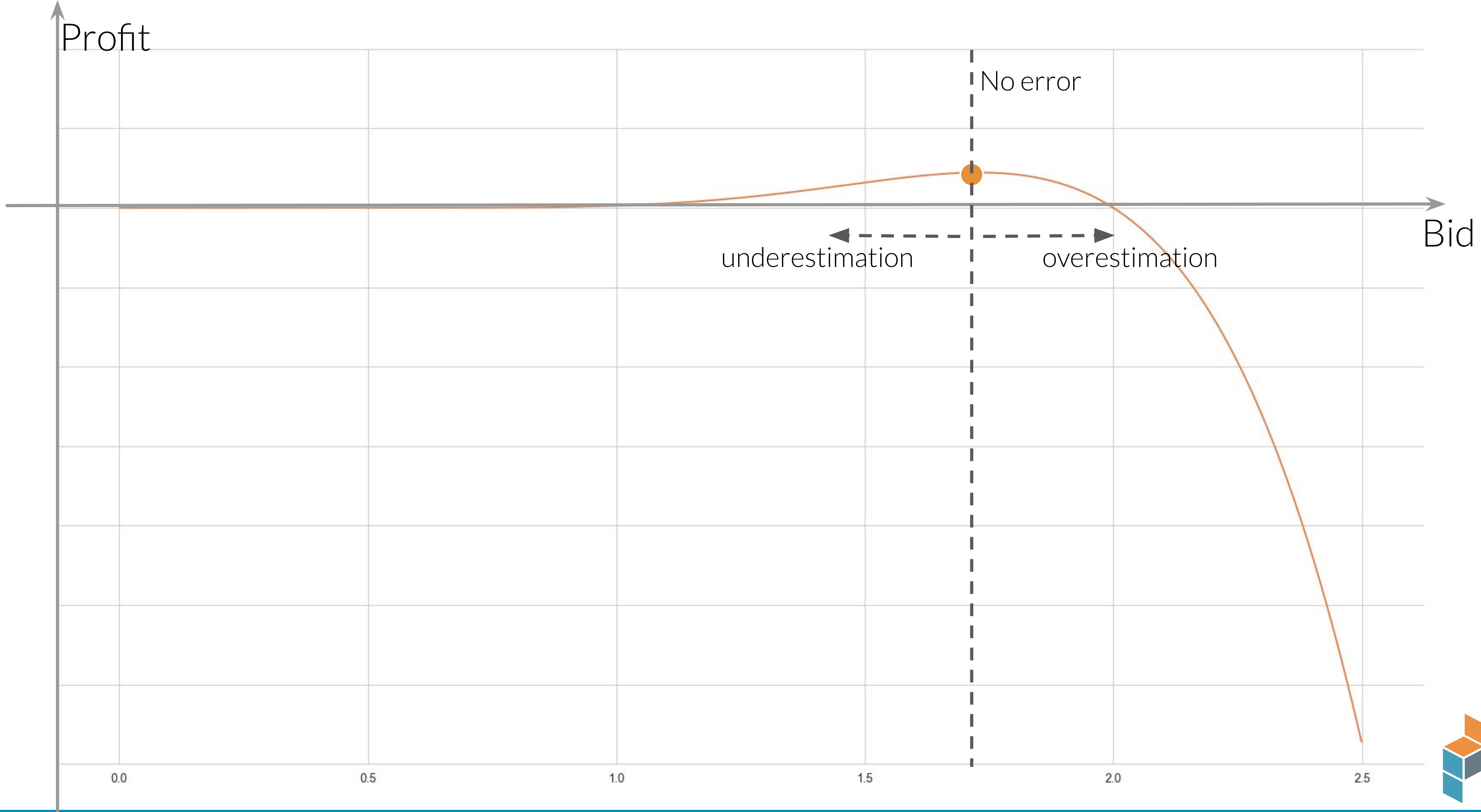
$$\underset{Bid}{\arg\max}\,((LTV - Bid) * Vol(Bid))$$


Profit
Bid
0.0
0.5
1.0
1.5
2.0
2.5

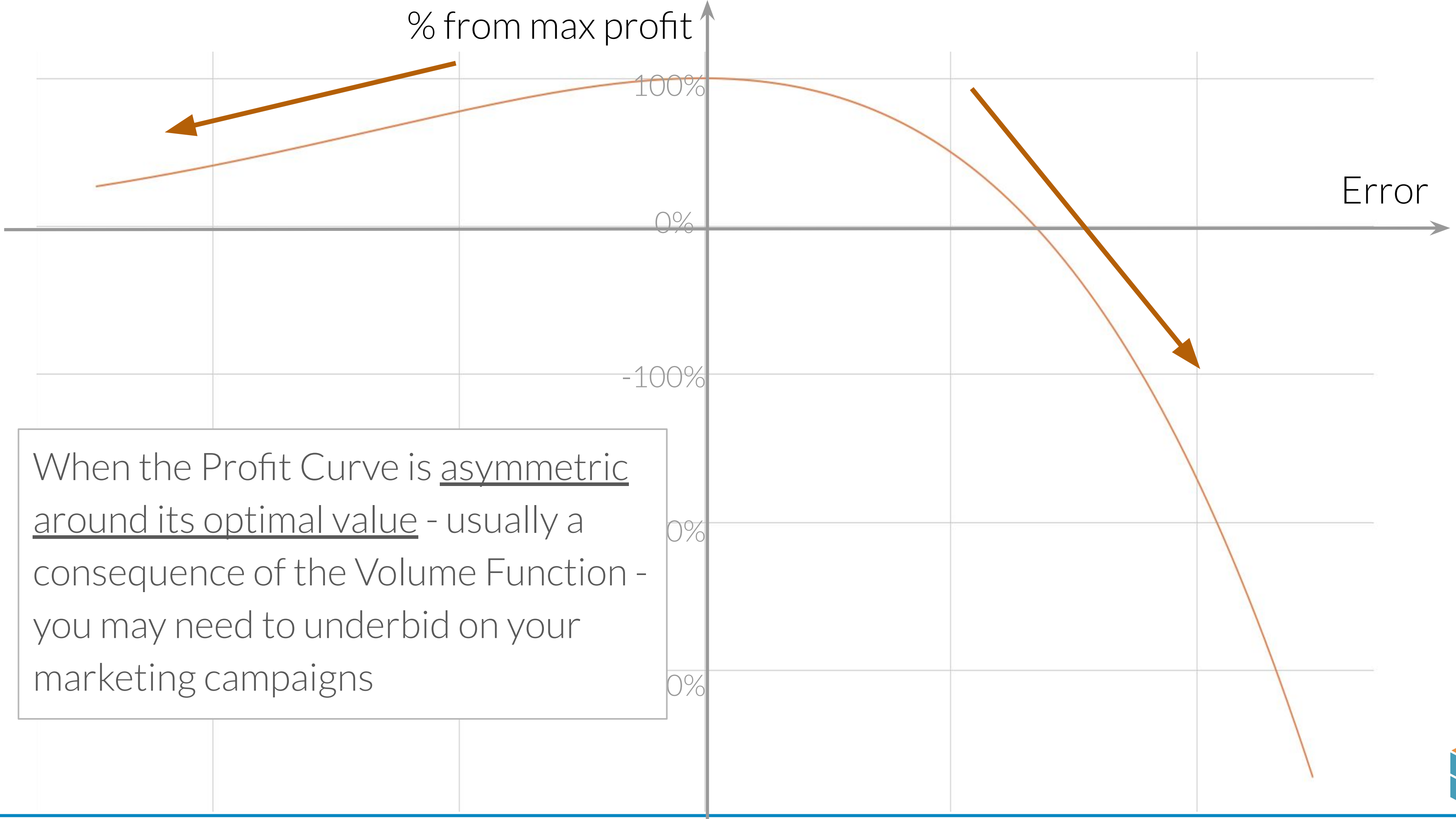

Profit
Real Profit Curve
Bid
Estimated Profit Curve
(Underestimated LTV)
0.0
0.5
1.0
1.5
2.0
2.5


Profit
Estimated Profit Curve
(Overestimated LTV)
Bid
Real Profit Curve
0.0
0.5
1.0
1.5
2.0
2.5

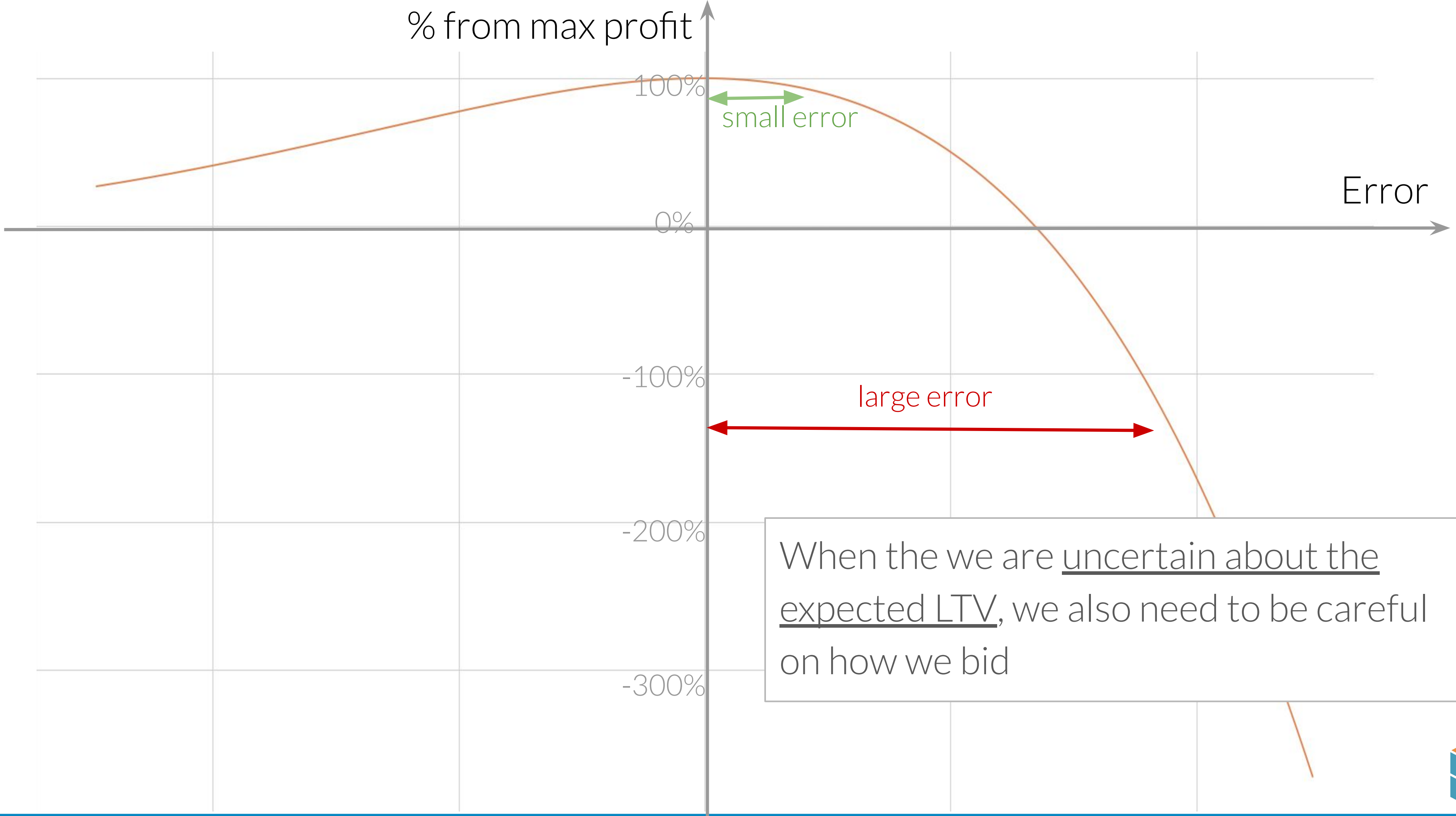

Profit
Estimated Profit Curve
(Overestimated LTV)
Estimated Profit
Real 'Profit'
Bid
Real Profit Curve
0.0
0.5
1.0
1.5
2.0
2.5


Profit
No error
Bid
underestimation
overestimation
0.0
0.5
1.0
1.5
2.0
2.5


% from max profit
Error
100%
0%
-100%
-200%
-300%



When the Profit Curve is asymmetric around its optimal value - usually a consequence of the Volume Function - you may need to underbid on your marketing campaigns



When the we are uncertain about the expected LTV, we also need to be careful on how we bid

Topics

- Introduction to Advertisement
  - Modelling advertisement for digital products
  - Finding the optimal bid for your marketing campaigns
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - **What is PySTAN**
  - The data used: Lifetime Value dataset from Kaggle
  - Description of the model
  - Implementation of the model in PySTAN
  - Achieving the same model with PyMC
  - Comparison between PySTAN and PyMC

What is (Py)STAN?

PySTAN is a Python interface to STAN.

STAN itself is a platform statistical modelling and allows high-performance statistical computation. It was first released in 2012, and since then has seen wide use in different scientific fields because of its interface to both R and Python, the implementation of efficient sampling algorithms, and usage of automatic differentiation

Topics

- Introduction to Advertisement
    - Modelling advertisement for digital products
    - Finding the optimal bid for your marketing campaigns
    - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
    - What is PySTAN
    - **The data used: Lifetime Value dataset from Kaggle**
    - Description of the model
    - Implementation of the model in PySTAN
    - Achieving the same model with PyMC
    - Comparison between PySTAN and PyMC

The dataset used: Lifetime Value from Kaggle

kaggle

Create

- Home
- Competitions
- Datasets
- Models
- Code
- Discussions
- Learn
- More

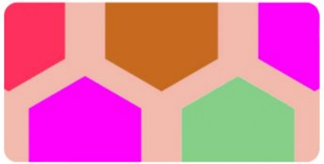
Search

Sign In | Register

DAVID RODRÍGUEZ SEGADO · UPDATED 3 YEARS AGO

9 | New Notebook | Download (11 MB)

Lifetime Value



Data Card | Code (2) | Discussion (0)

About Dataset

Explanation of the data set:

The files contain randomized data about the behaviour and spending patterns of customers. This data is used to model the amount a user will spend during his membership on specific websites. This metric is called customer lifetime value (LTV). The training set data are from users that have joined our website from Dec/2018 through June/2019. The test set contains users that joined in July/2019.
The data set contains the following columns:

- **product_type**: one of 4 different product_type a user can purchase. The user always gets a first product (called 'type_p') and then can purchase additional products (of 'type_x', 'type_u', and 'type_ex').
- **user_id**: a numerical identifier of the user. Note that for every user we might

Usability
5.59

License
Unknown

Expected update frequency
Not specified

Tags
Business

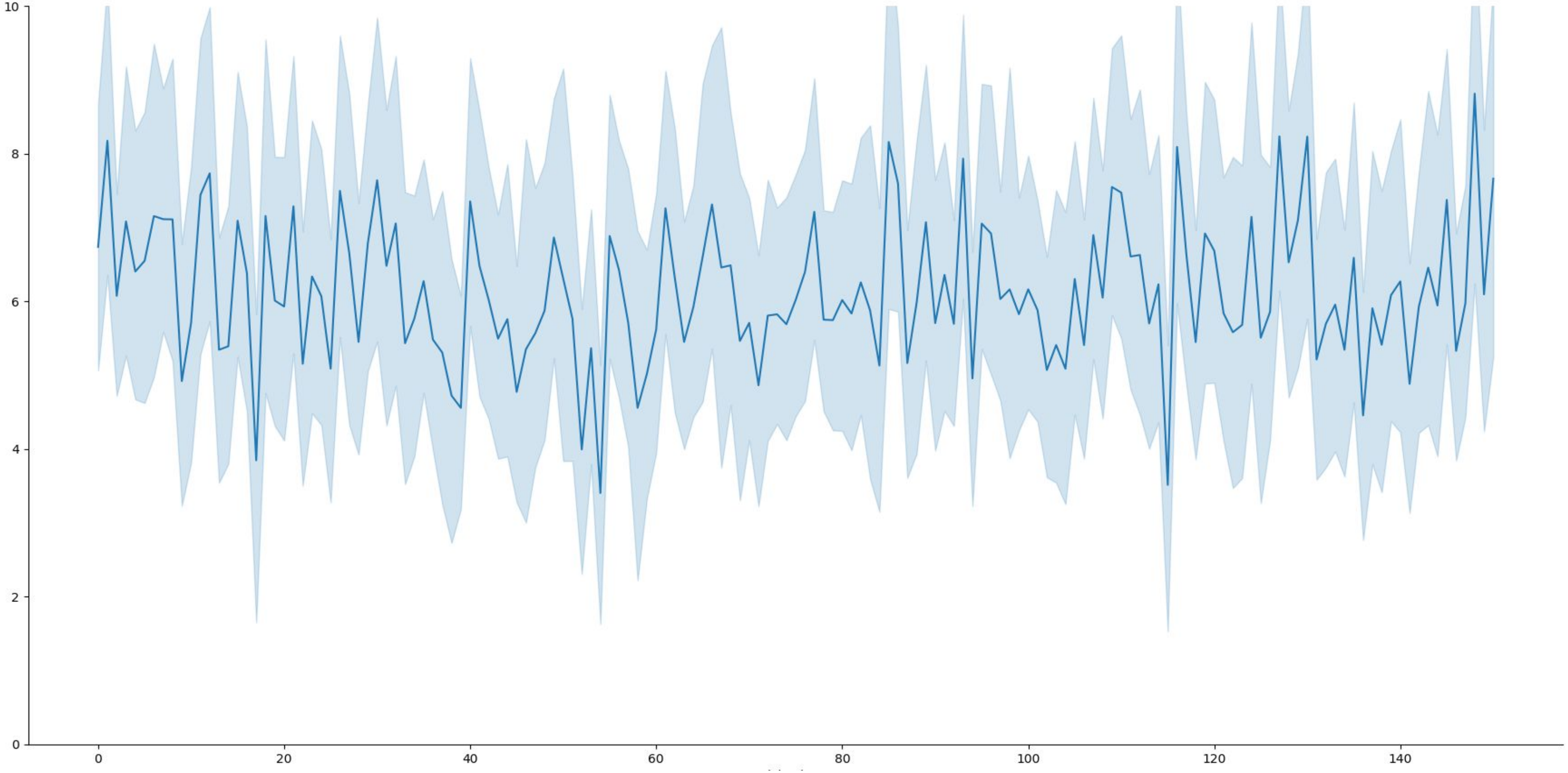
The dataset used: Lifetime Value from Kaggle

	product_type	user_id	join_date	hidden	product	STV	target	credit_card_level	is_lp	aff_type	is_cancelled	country_segment
0	type_ex	7.0	2018-12-01 00:01:45	0	product_1	8.25	8.25	standard	0	PPL	NaN	US
1	type_ex	20.0	2018-12-01 00:06:05	0	product_2	8.25	8.25	standard	0	PPL	NaN	US
2	type_ex	22.0	2018-12-01 00:06:23	0	product_3	8.25	8.25	prepaid	0	PPL	NaN	US
3	type_ex	26.0	2018-12-01 00:07:12	0	product_2	8.25	8.25	standard	0	PPL	NaN	US
4	type_ex	59.0	2018-12-01 00:15:21	0	product_2	8.25	8.25	standard	0	PPL	NaN	Other Countries
5	type_ex	63.0	2018-12-01 00:15:28	0	product_1	8.25	8.25	standard	0	PPS	NaN	US
6	type_ex	70.0	2018-12-01 00:16:32	0	product_4	8.25	8.25	standard	0	PPL	NaN	US
7	type_ex	82.0	2018-12-01 00:21:14	0	product_1	8.25	8.25	standard	0	PPS	NaN	US
8	type_ex	87.0	2018-12-01 00:22:40	0	product_2	8.25	8.25	standard	1	PPS	NaN	Other Countries
9	type_ex	102.0	2018-12-01 00:26:55	0	product_2	8.25	8.25	standard	0	PPL	NaN	US

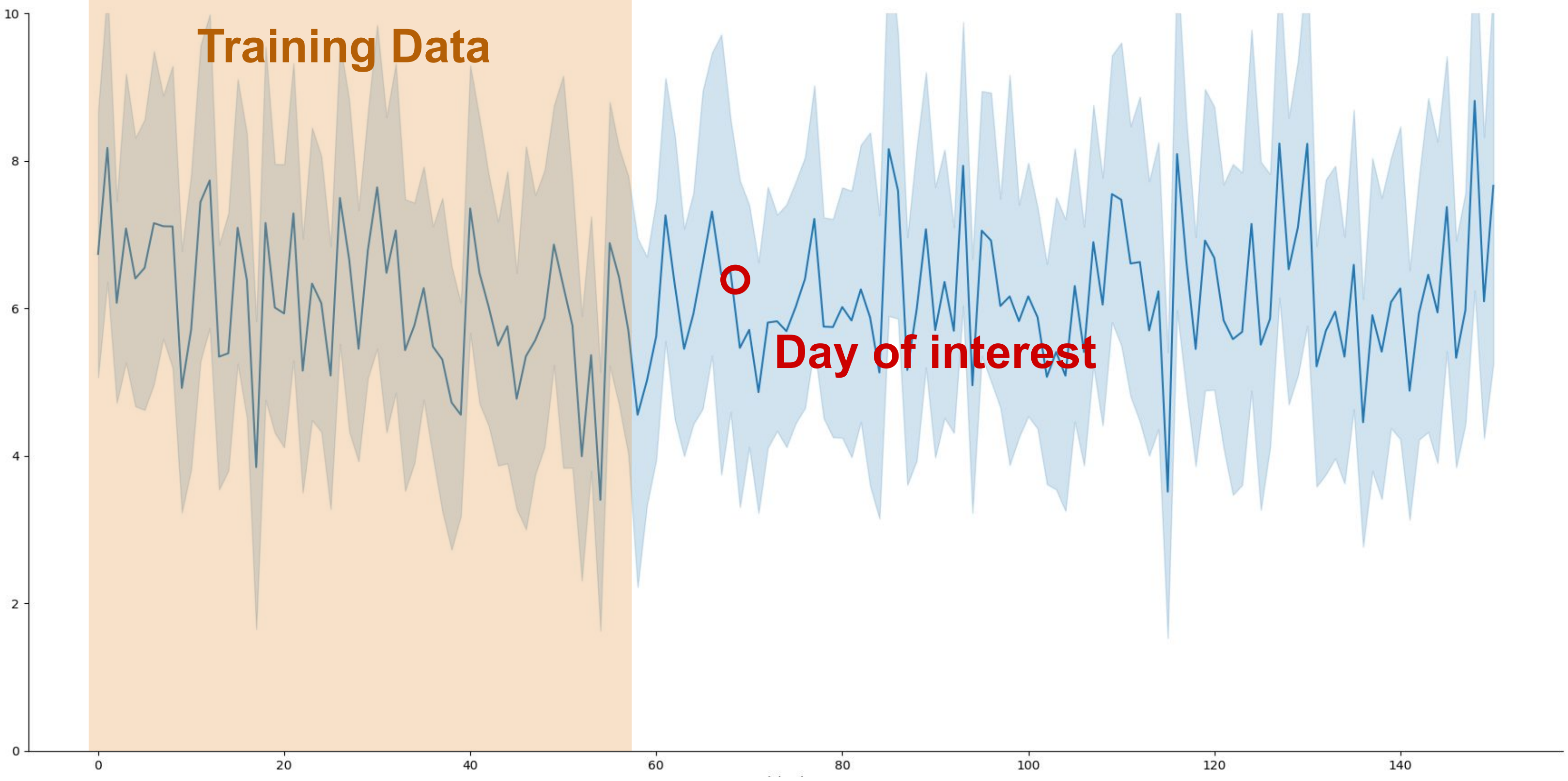
The dataset used: Lifetime Value from Kaggle

	product_type	user_id	join_date	hidden	product	STV	target	credit_card_level	is_lp	aff_type	is_cancelled	country_segment
0	type_ex	7.0	2018-12-01 00:01:45	0	product_1	8.25	8.25	standard	0	PPL	NaN	US
1	type_ex	20.0	2018-12-01 00:06:05	0	product_2	8.25	8.25	standard	0	PPL	NaN	US
2	type_ex	22.0	2018-12-01 00:06:23	0	product_3	8.25	8.25	prepaid	0	PPL	NaN	US
3	type_ex	26.0	2018-12-01 00:07:12	0	product_2	8.25	8.25	standard	0	PPL	NaN	US
4	type_ex	59.0	2018-12-01 00:15:21	0	product_2	8.25	8.25	standard	0	PPL	NaN	Other Countries
5	type_ex	63.0	2018-12-01 00:15:28	0	product_1	8.25	8.25	standard	0	PPS	NaN	US
6	type_ex	70.0	2018-12-01 00:16:32	0	product_4	8.25	8.25	standard	0	PPL	NaN	US
7	type_ex	82.0	2018-12-01 00:21:14	0	product_1	8.25	8.25	standard	0	PPS	NaN	US
8	type_ex	87.0	2018-12-01 00:22:40	0	product_2	8.25	8.25	standard	1	PPS	NaN	Other Countries
9	type_ex	102.0	2018-12-01 00:26:55	0	product_2	8.25	8.25	standard	0	PPL	NaN	US

The dataset used: Lifetime Value from Kaggle



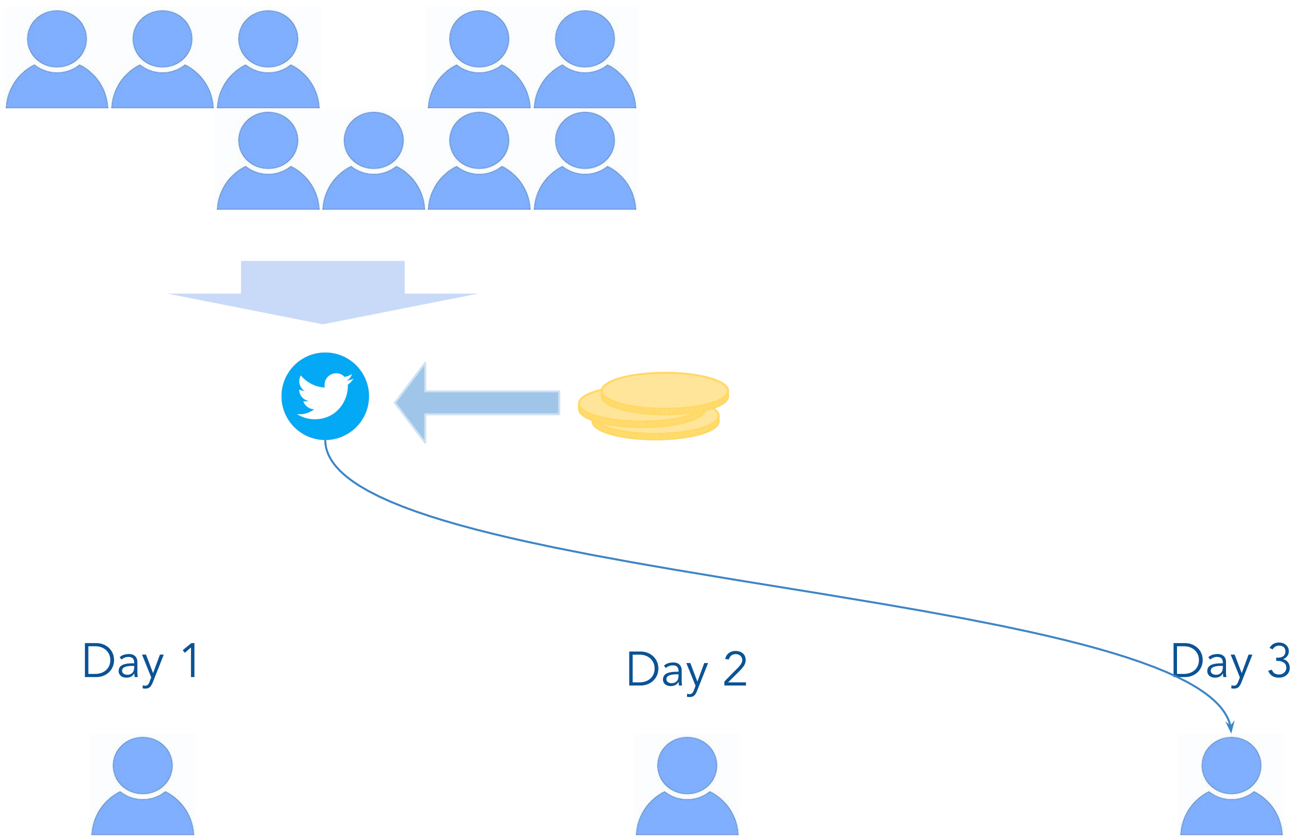
The dataset used: Lifetime Value from Kaggle

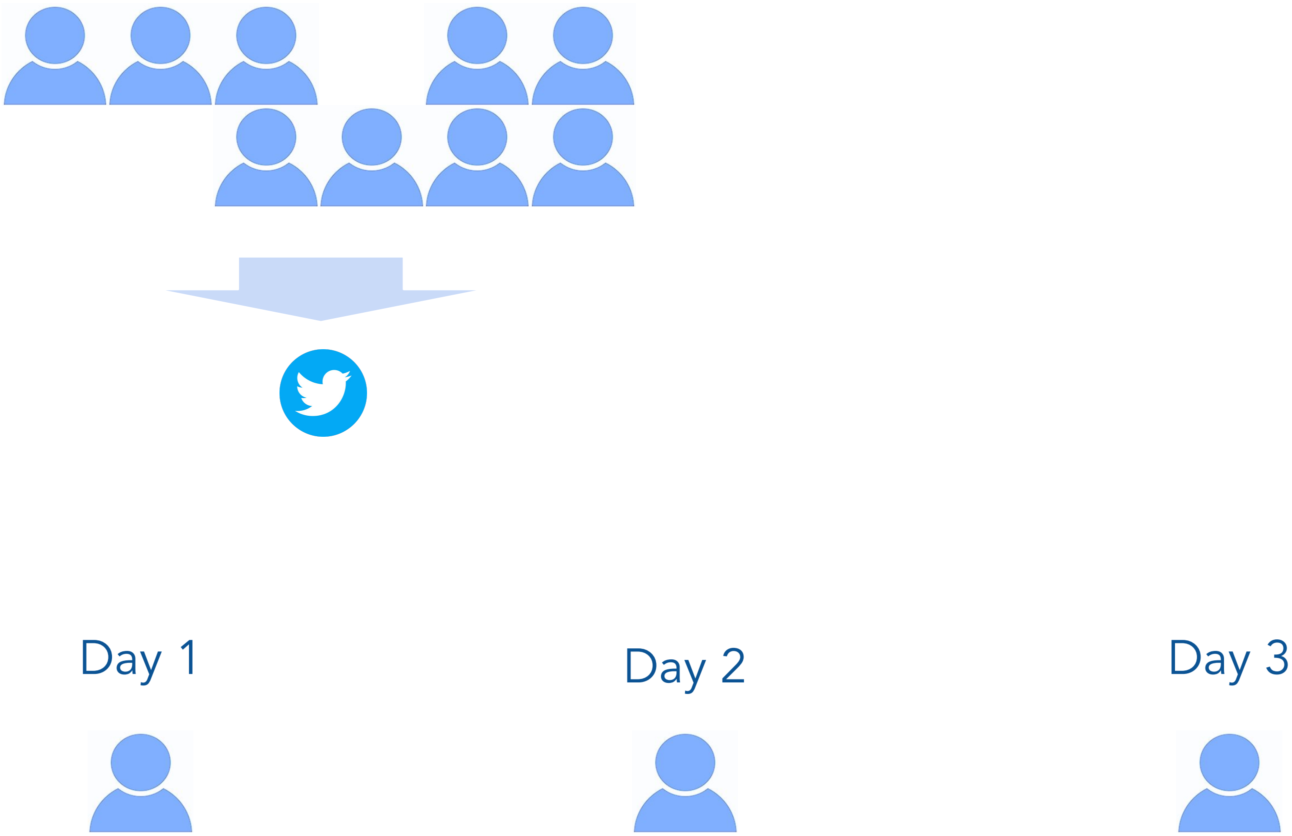


Topics

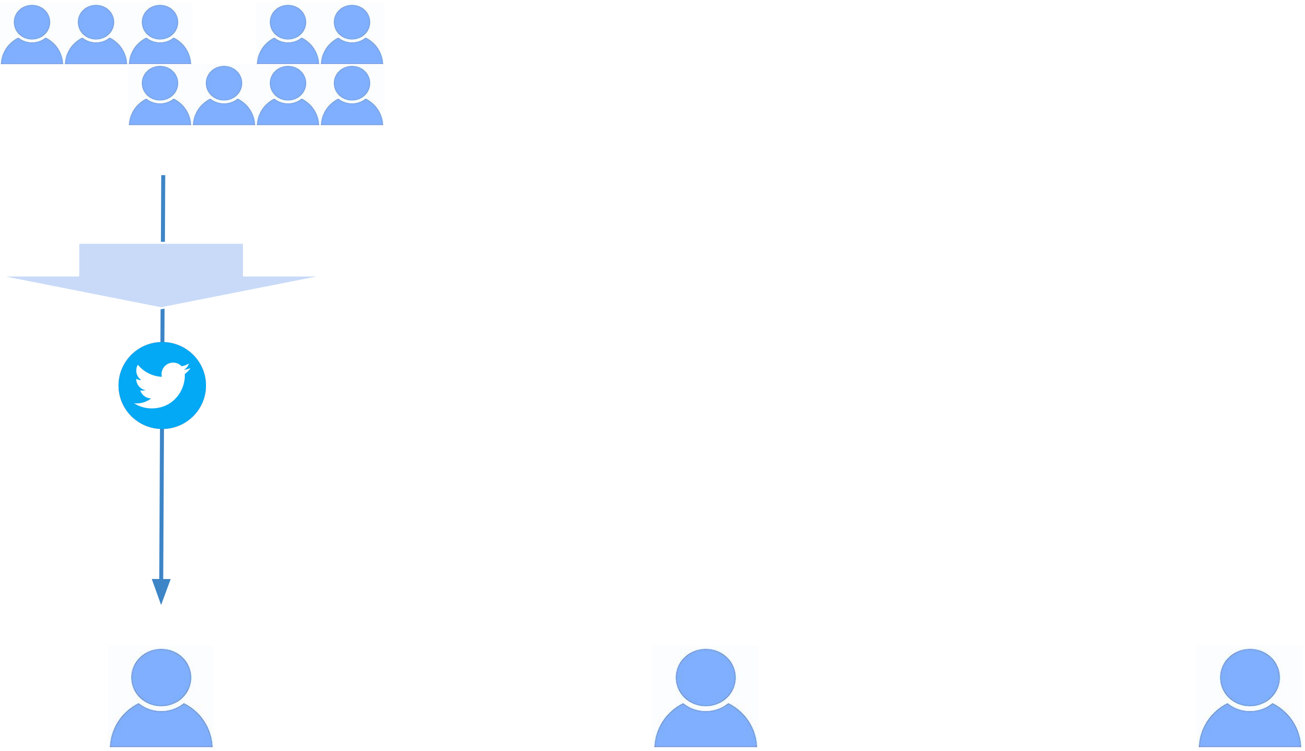
- Introduction to Advertisement
    - Modelling advertisement for digital products
    - Finding the optimal bid for your marketing campaigns
    - The role that uncertainty play in the marketing strategy

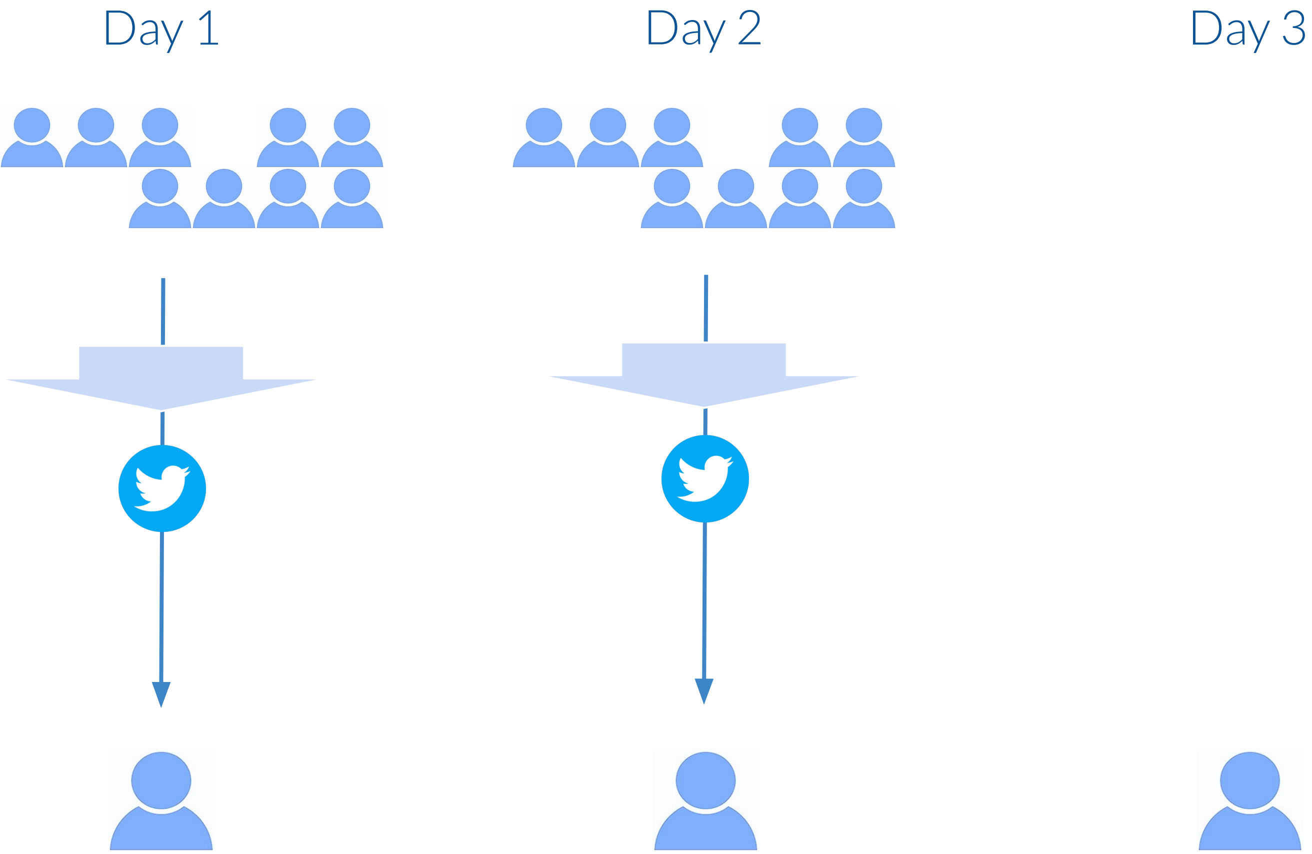
- Forecasting Lifetime Value with PySTAN
    - What is PySTAN
    - The data used: Lifetime Value dataset from Kaggle
    - **Description of the model**
    - Implementation of the model in PySTAN
    - Achieving the same model with PyMC
    - Comparison between PySTAN and PyMC

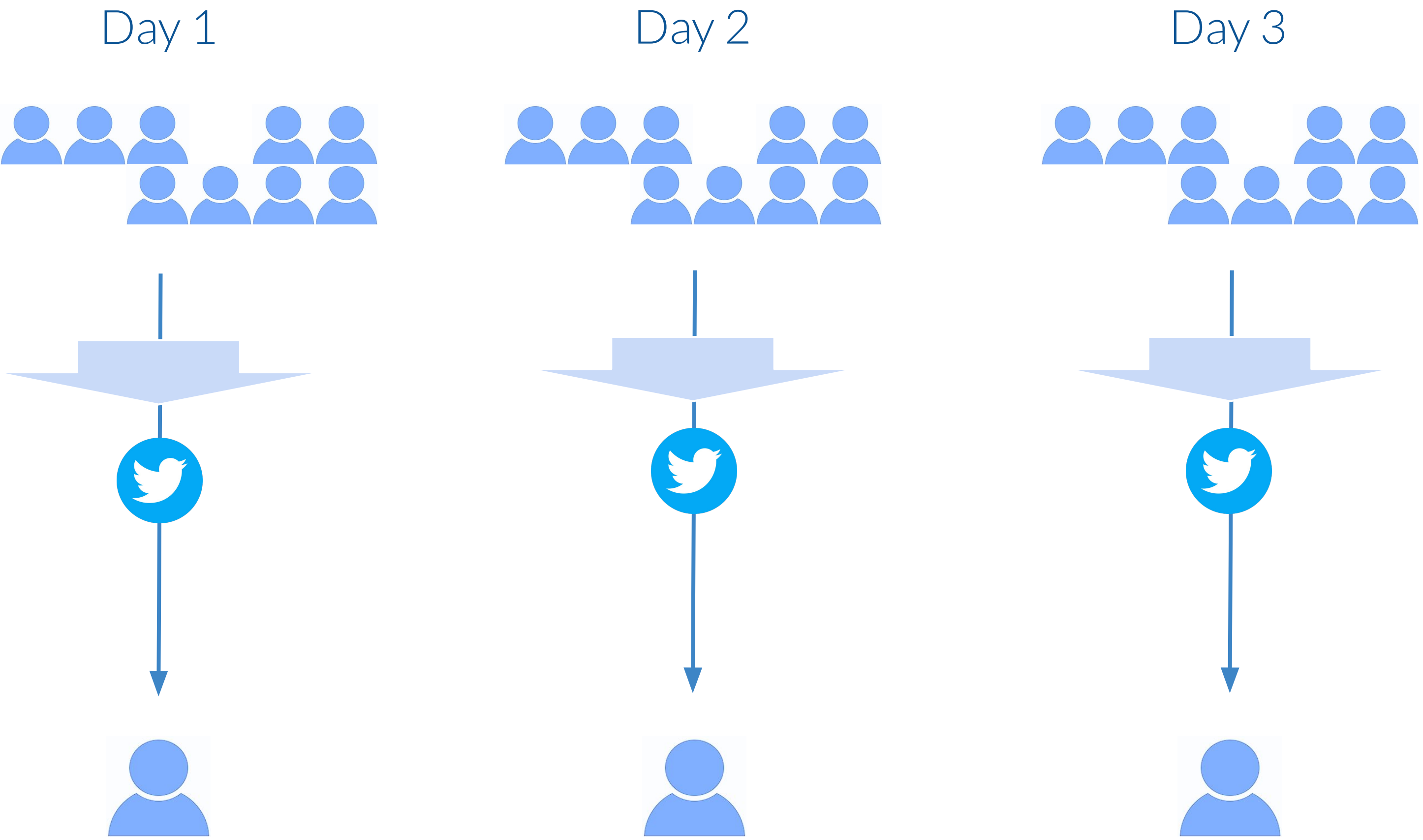

Day 1
Day 2
Day 3

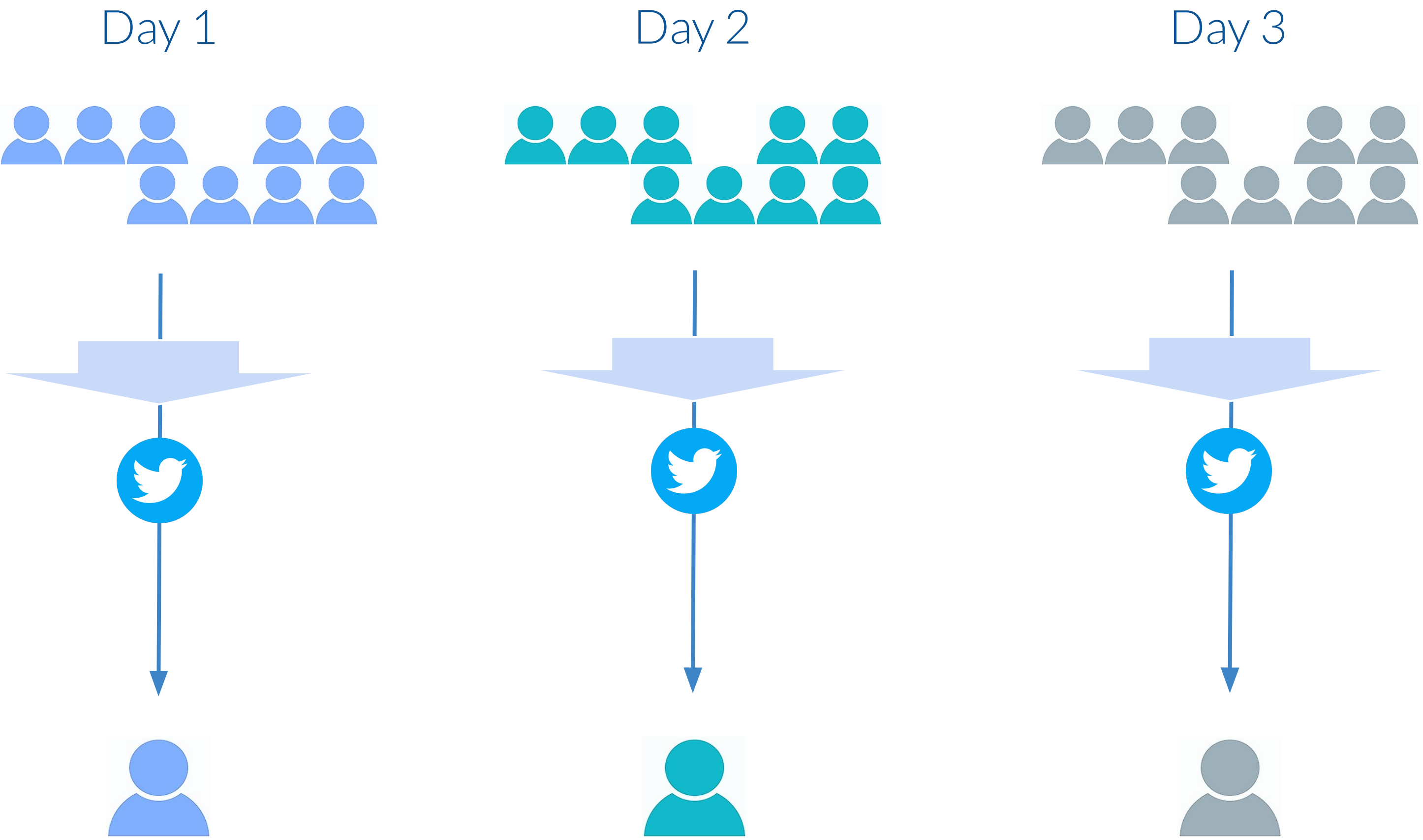

Day 1
Day 2
Day 3

Day 1 Day 2 Day 3

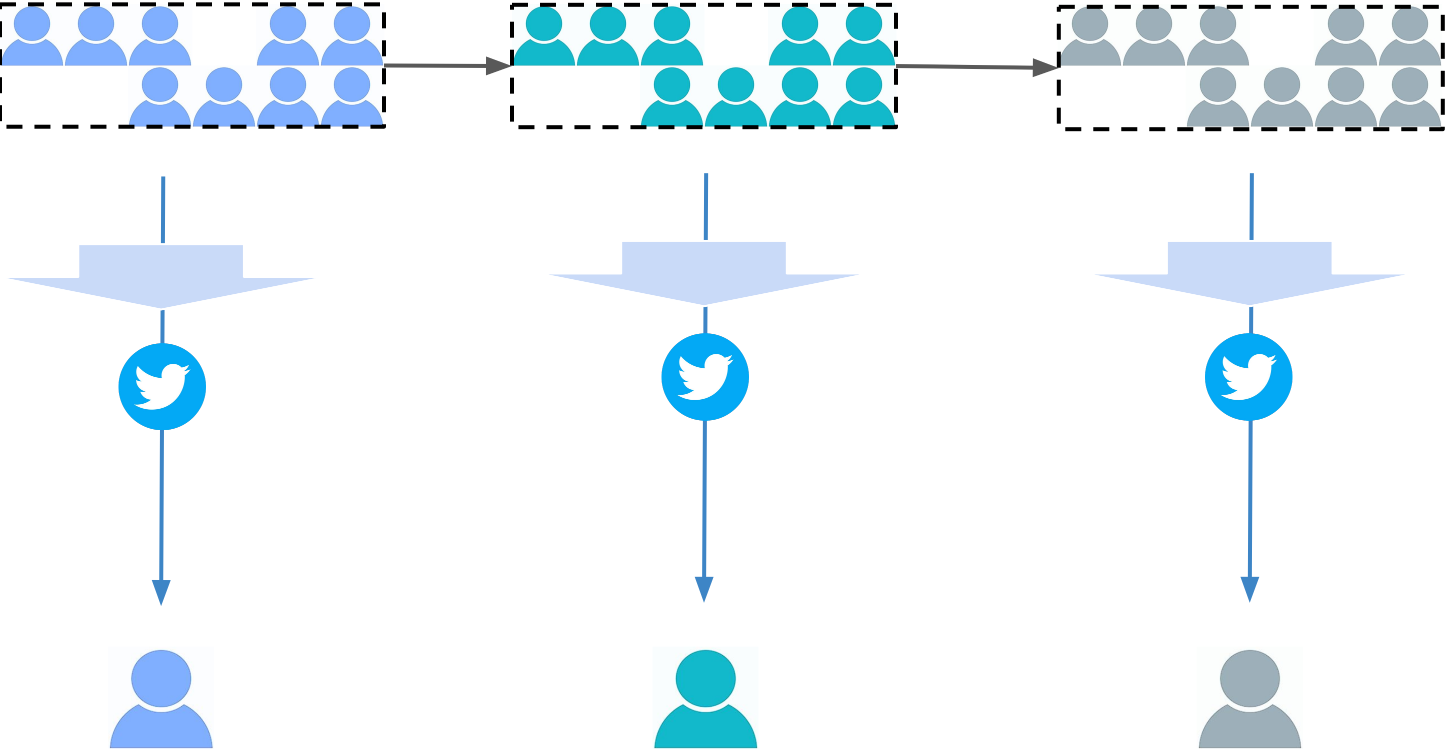



Day 1
Day 2
Day 3


Day 1
Day 2
Day 3

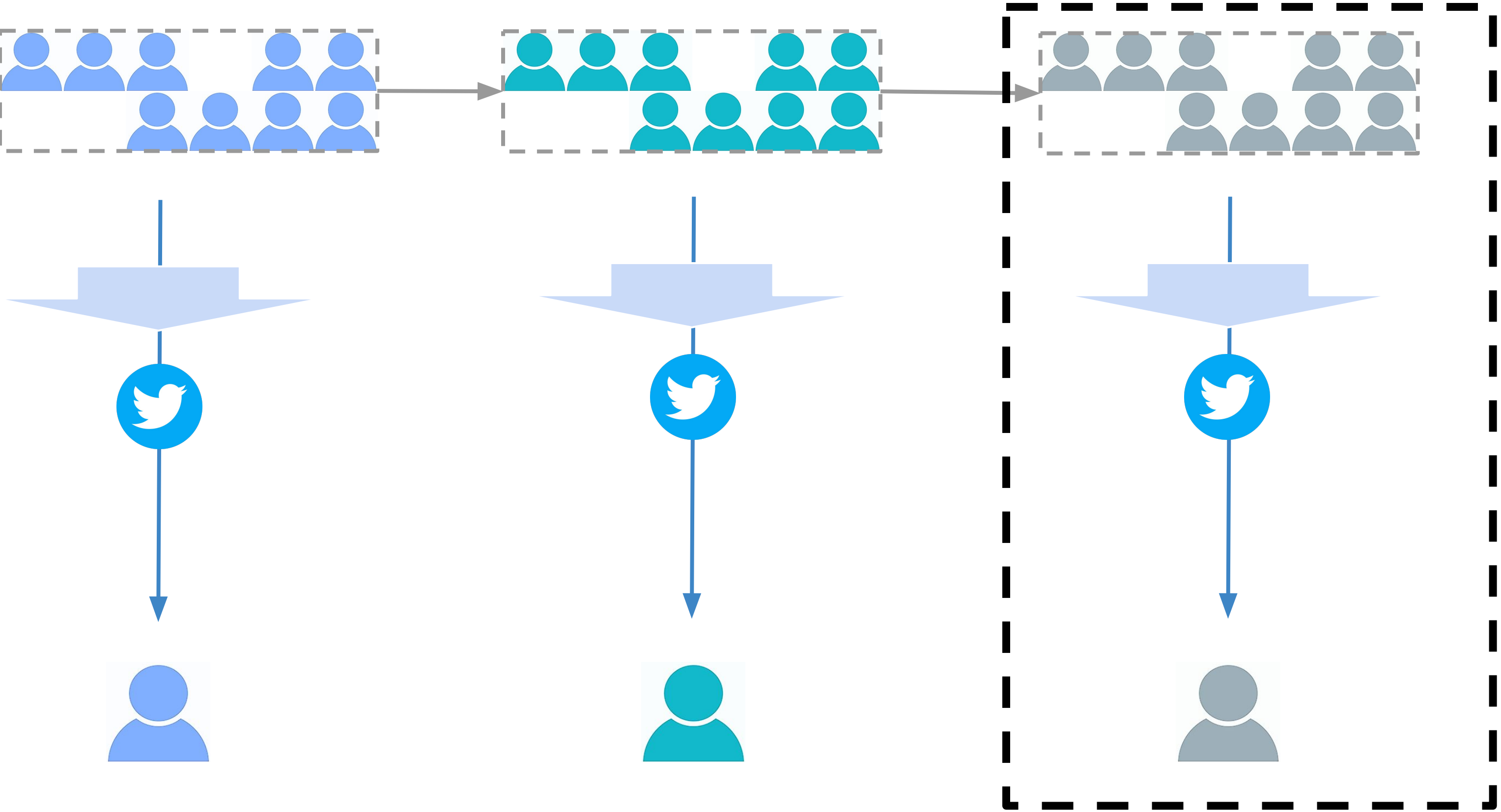

Day 1
Day 2
Day 3

$$LTV_{Population}(t) \sim Normal\left(LTV_{Population}(t-1), \sigma^2\right)$$



Random Walk

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$



Sampling Process

The (Py)STAN Model

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

Modelling population LTV as a Random Walk

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

LTV of a user is a ‘sample’ of the population LTV

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

Prior for population LTV

Topics

- Introduction to Advertisement
  - Modelling advertisement for digital products
  - Finding the optimal bid for your marketing campaigns
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - What is PySTAN
  - The data used: Lifetime Value dataset from Kaggle
  - Description of the model
  - **Implementation of the model in PySTAN**
  - Achieving the same model with PyMC
  - Comparison between PySTAN and PyMC

The (Py)STAN Model

```
import stan
                     str                     dict
model = stan.build(time_series_model, data=stan_data)
posterior_distributions =  model.sample(numchains=3, num_samples=1000)
```

The (Py)STAN Model

```
import stan

model = stan.build(time_series_model, data=stan_data)
posterior_distributions =  model.sample(numchains=3, num_samples=1000)
```

```
data = pd.read_csv('lifetime_value.csv')

stan_data = {}
stan_data['N'] = data.shape['0']
stan_data['n_dates'] = len(data['join_date'].unique())
stan_data['date'] = data['join_date']
stan_data['observation'] = data['target']
```

The (Py)STAN Model

```
import stan

model = stan.build(time_series_model, data=stan_data)
posterior_distributions =  model.sample(numchains=3, num_samples=1000)
```

The (Py)STAN Model

```
time_series_model = “”””
        data {




        }
        parameters {



        }
        model {






        }
“”””
```

The (Py)STAN Model

```
time_series_model = “”””
      data {



      }
      parameters {



      }
      model {






      }
“”””
```

Data defines what are the data and constants passed to the model

Parameters declare the variables you want to estimate

Model defines the connection between the parameters and the data

The (Py)STAN Model

```
time_series_model = “”””
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {



      }
      model {







      }
“”””
```

```
data = pd.read_csv(‘lifetime_value.csv’)

stan_data = {}
stan_data[‘N’] = data.shape[‘0’]
stan_data[‘n_dates’] = len(data[‘join_date’].unique())
stan_data[‘date’] = data[‘join_date’]
stan_data[‘observation’] = data[‘target’]
```

The (Py)STAN Model

```
time_series_model = “”””
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {



      }
      model {







      }
“”””
```

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

The (Py)STAN Model

```
time_series_model = """
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {
          vector[n_dates] mu;


      }
      model {






      }
"""
```

$$\boxed{LTV_{Population}(t)} \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

The (Py)STAN Model

```
time_series_model = “””
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {
          vector[n_dates] mu;
          real<lower=0.0001> sampling_stddev;

      }
      model {





      }
“””
```

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \boxed{\sigma^2_{Sampling}})$$

The (Py)STAN Model

```
time_series_model = “””
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {
          vector[n_dates] mu;
          real<lower=0.0001> sampling_stddev;
          real<lower=0.0001> random_walk_stddev;
      }
      model {



      }
“””
```

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \boxed{\sigma^2_{Time}})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

The (Py)STAN Model

```
time_series_model = """
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {
          vector[n_dates] mu;
          real<lower=0.0001> sampling_stddev;
          real<lower=0.0001> random_walk_stddev;
      }
      model {
          sampling_stddev ~ cauchy(0, 2);
          random_walk_stddev ~ cauchy(0, 2);
          mu[1] ~ normal(6, 3);



      }
"""
```

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$\boxed{LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})}$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

The (Py)STAN Model

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

```
time_series_model = """
    data {
      int<lower=0> N;
      int<lower=0> n_dates;
      int date[N];
      real observation[N];
    }
    parameters {
        vector[n_dates] mu;
        real<lower=0.0001> sampling_stddev;
        real<lower=0.0001> random_walk_stddev;
    }
    model {
        sampling_stddev ~ cauchy(0, 2);
        random_walk_stddev ~ cauchy(0, 2);
        mu[1] ~ normal(6, 3);
        mu[2:n_dates] ~ normal(mu[1:(n_dates - 1)], random_walk_stddev);


    }
"""
```

The (Py)STAN Model

```
time_series_model = “””
      data {
        int<lower=0> N;
        int<lower=0> n_dates;
        int date[N];
        real observation[N];
      }
      parameters {
          vector[n_dates] mu;
          real<lower=0.0001> sampling_stddev;
          real<lower=0.0001> random_walk_stddev;
      }
      model {
          sampling_stddev ~ cauchy(0, 2);
          random_walk_stddev ~ cauchy(0, 2);
          mu[1] ~ normal(6, 3);
          mu[2:n_dates] ~ normal(mu[1:(n_dates - 1)], random_walk_stddev);

          observation ~ normal(mu[date], sampling_stddev);
      }
“””
```

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

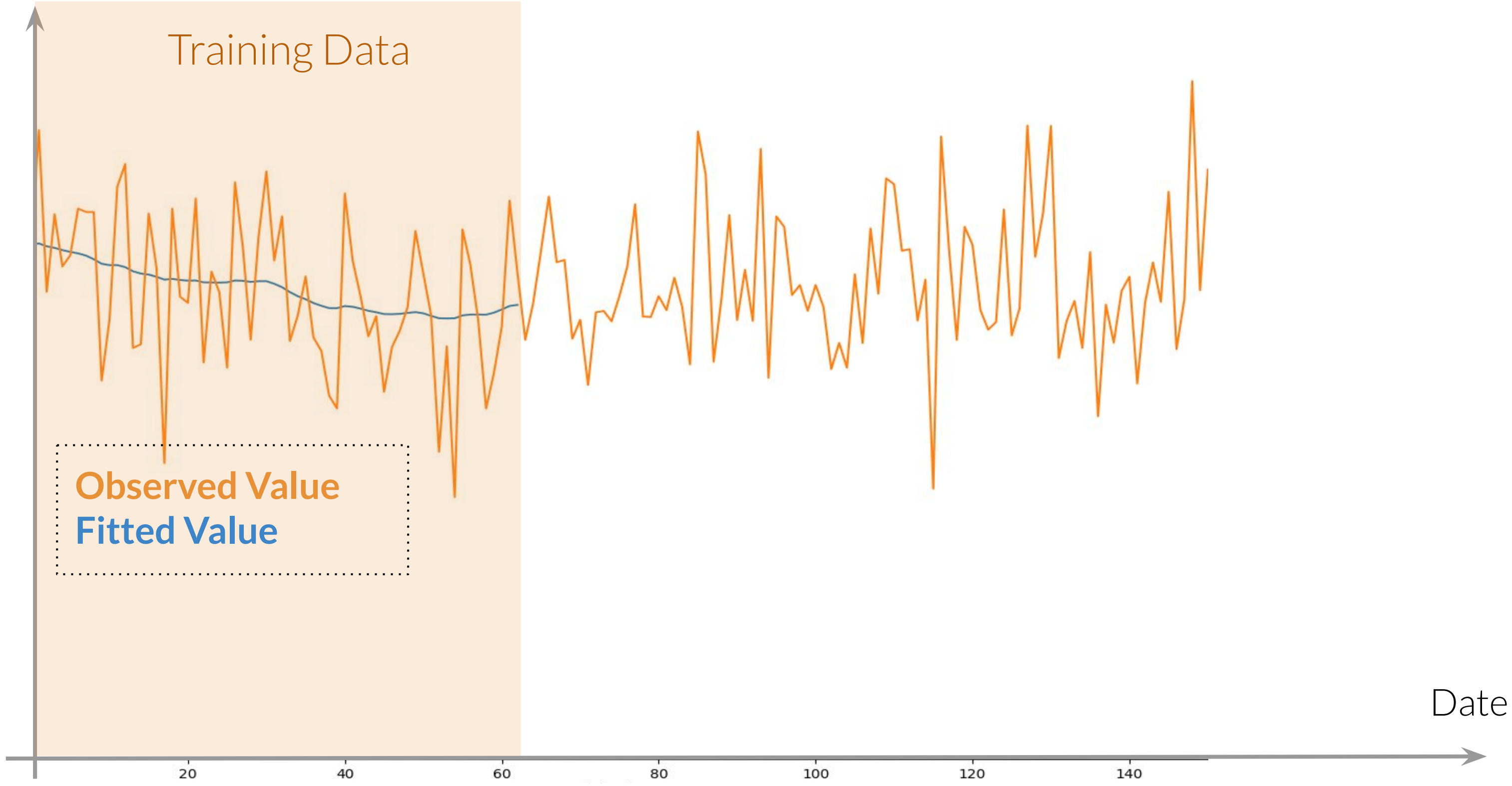
$$\boxed{LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})}$$

The (Py)STAN Model

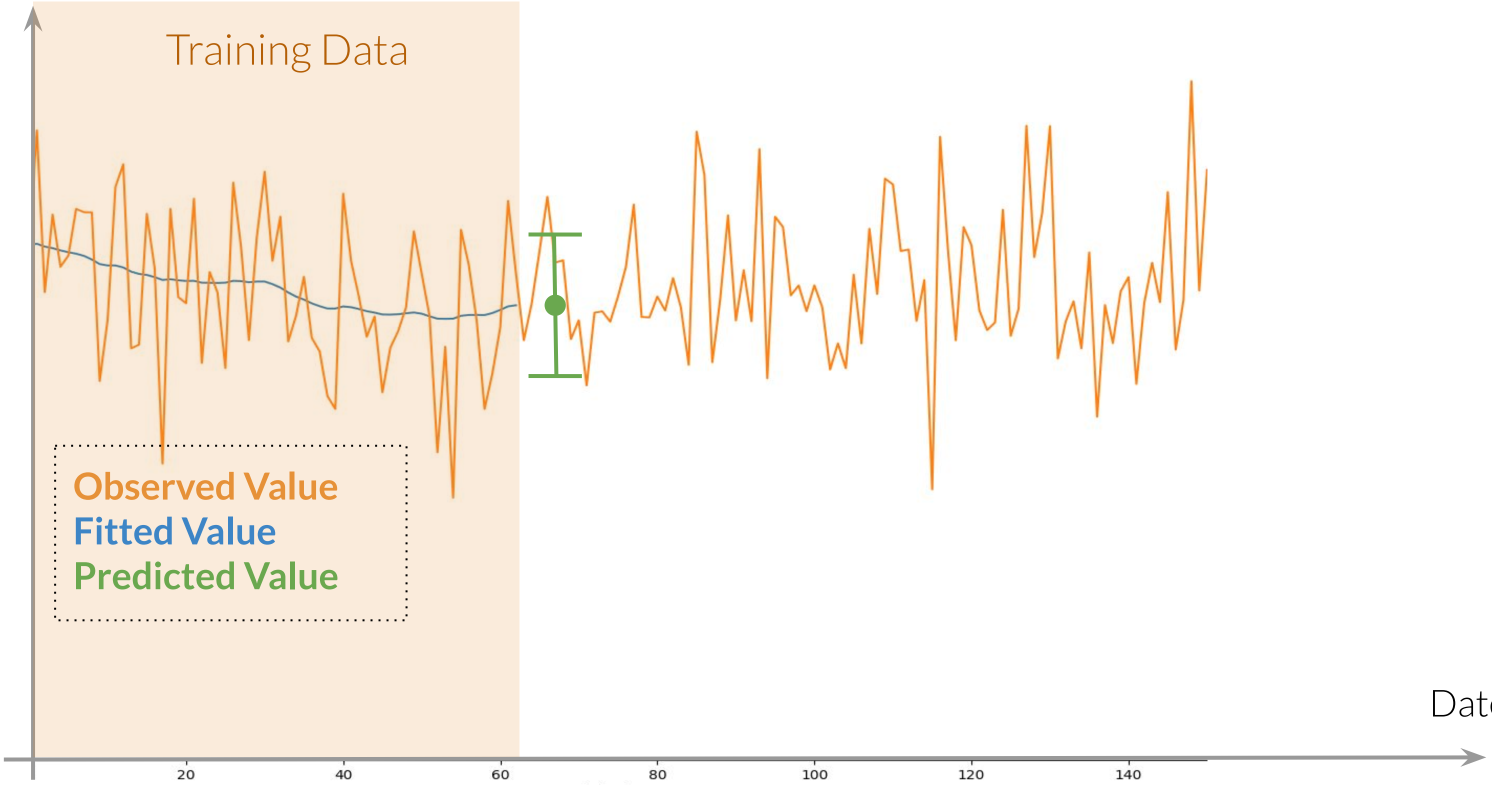
```
import stan

model = stan.build(time_series_model, data=stan_data)
posterior_distributions =  model.sample(numchains=3, num_samples=1000)
```

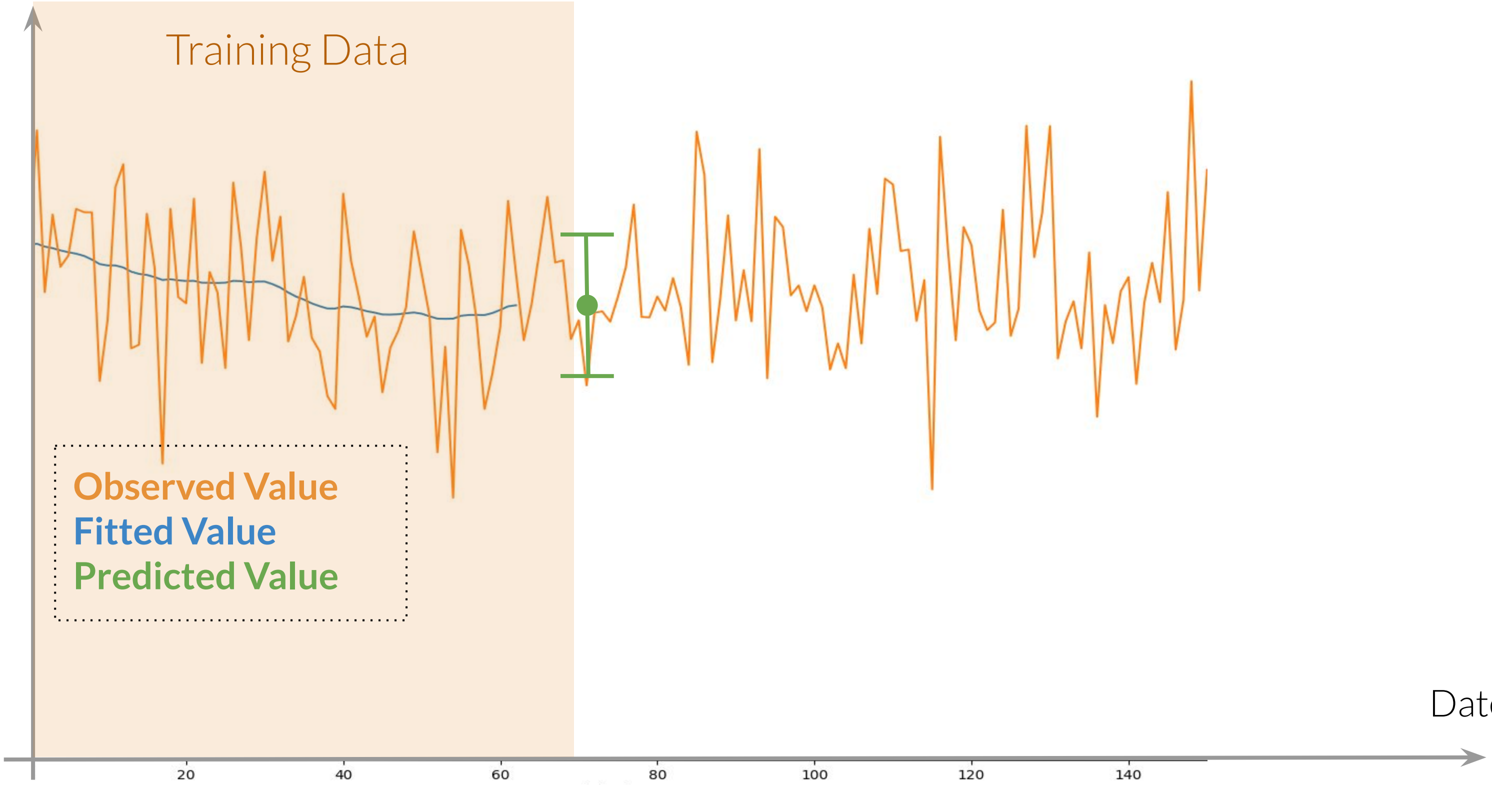
The (Py)STAN Model



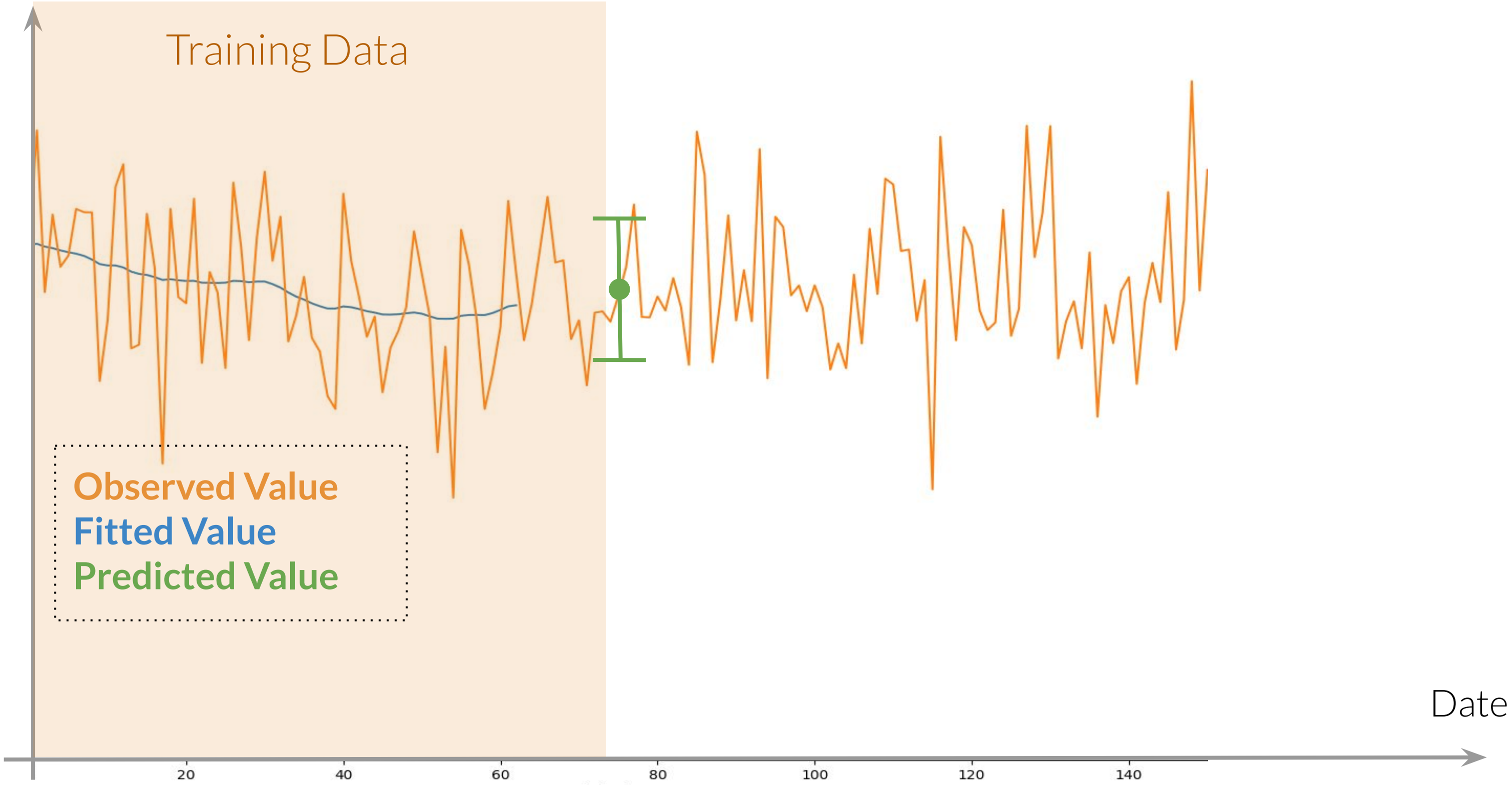
The (Py)STAN Model



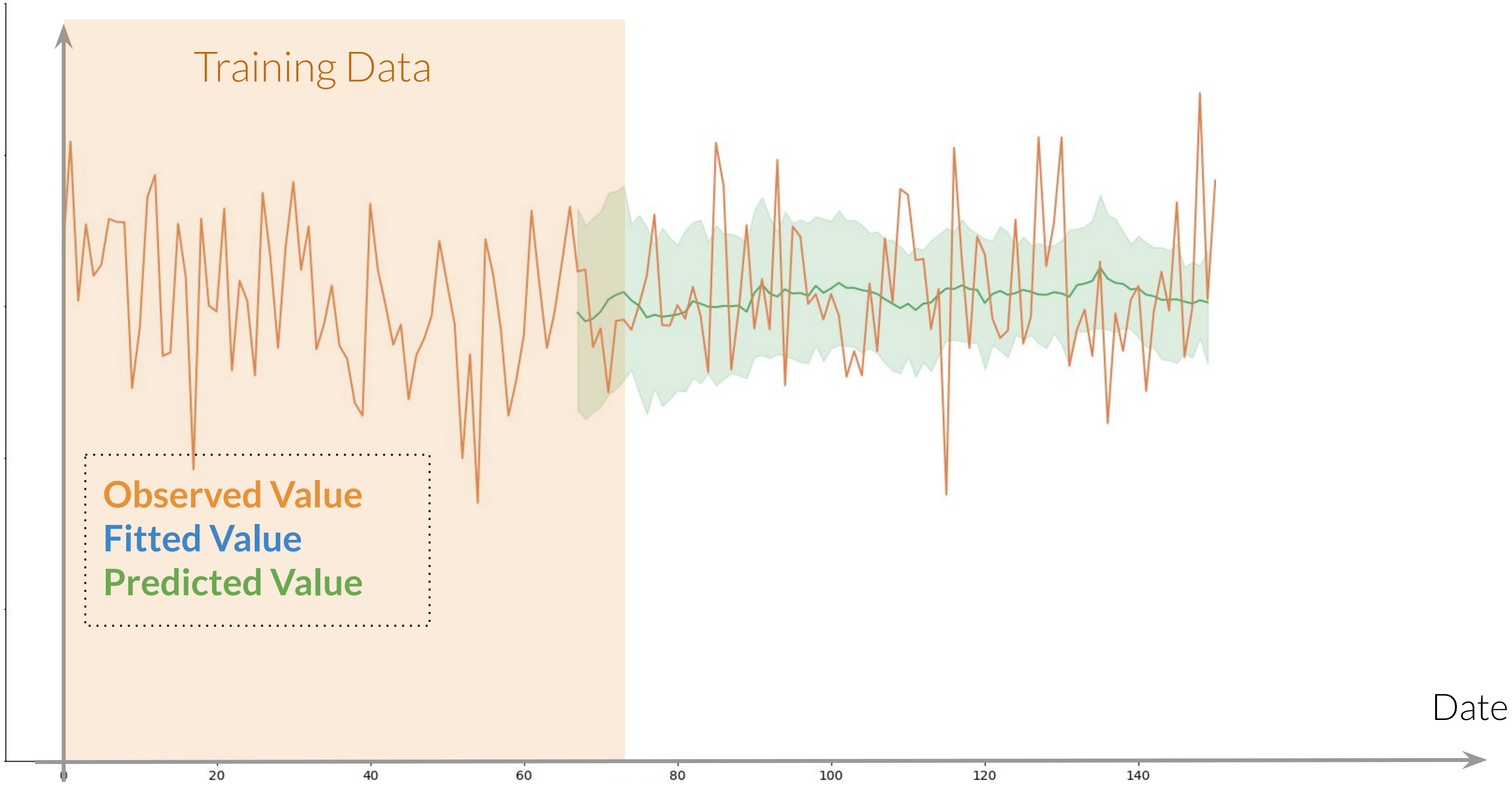
The (Py)STAN Model



The (Py)STAN Model



The (Py)STAN Model



Topics

- Introduction to Advertisement
  - Modelling advertisement for digital products
  - Finding the optimal bid for your marketing campaigns
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - What is PySTAN
  - Description of the model
  - The data used: Lifetime Value dataset from Kaggle
  - Implementation of the model in PySTAN
  - **Achieving the same model with PyMC**
  - Comparison between PySTAN and PyMC

The PyMC Model

```
import pymc as pm

coords = {"steps": data['join_date'].values}
with pm.Model(coords=coords) as model:
```

The PyMC Model

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$\boxed{LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})}$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

```
import pymc as pm

coords = {"steps": data['join_date'].values}
with pm.Model(coords=coords) as model:

    # Priors on Gaussian random walks
    mu_prior = pm.Normal.dist(6, 3)
    random_walk_stddev = pm.HalfCauchy('random_walk_stddev', 2)
    sampling_stddev = pm.HalfCauchy("sampling_stddev", 2)
```

The PyMC Model

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})$$

```
import pymc as pm

coords = {"steps": data['join_date'].values}
with pm.Model(coords=coords) as model:

    # Priors on Gaussian random walks
    mu_prior = pm.Normal.dist(6, 3)
    random_walk_stddev = pm.HalfCauchy('random_walk_stddev', 2)
    sampling_stddev = pm.HalfCauchy("sampling_stddev", 2)

    alpha = pm.Deterministic(
      "alpha", pt.concatenate([alpha_prior,
      pm.Normal("alpha_raw", sigma=alpha_dev, shape=n_days-1)]).cumsum(axis=0)
      )
```

The PyMC Model

$$LTV_{Population}(t) \sim Normal(LTV_{Population}(t-1), \sigma^2_{Time})$$

$$LTV_{Population}(t=0) \sim Normal(\mu_{Prior}, \sigma^2_{Prior})$$

$$\boxed{LTV_{User}(t) \sim Normal(LTV_{Population}(t), \sigma^2_{Sampling})}$$

```
import pymc as pm

coords = {"steps": data['join_date'].values}
with pm.Model(coords=coords) as model:

    # Priors on Gaussian random walks
    mu_prior = pm.Normal.dist(6, 3)
    random_walk_stddev = pm.HalfCauchy('random_walk_stddev', 2)
    sampling_stddev = pm.HalfCauchy("sampling_stddev", 2)

    alpha = pm.Deterministic(
      "alpha", pt.concatenate([alpha_prior,
      pm.Normal("alpha_raw", sigma=alpha_dev, shape=n_days-1)]).cumsum(axis=0)
      )

    likelihood = pm.Normal(
      "likelihood", mu=mu[dates_tensor], sigma=sampling_stddev, observed=target_tensor, dims="steps"
    )
```

Topics

- Introduction to Advertisement
  - Modelling advertisement for digital products
  - Finding the optimal bid for your marketing campaigns
  - The role that uncertainty play in the marketing strategy

- Forecasting Lifetime Value with PySTAN
  - What is PySTAN
  - The data used: Lifetime Value dataset from Kaggle
  - Description of the model
  - Implementation of the model in PySTAN
  - Achieving the same model with PyMC
  - **Comparison between PySTAN and PyMC**

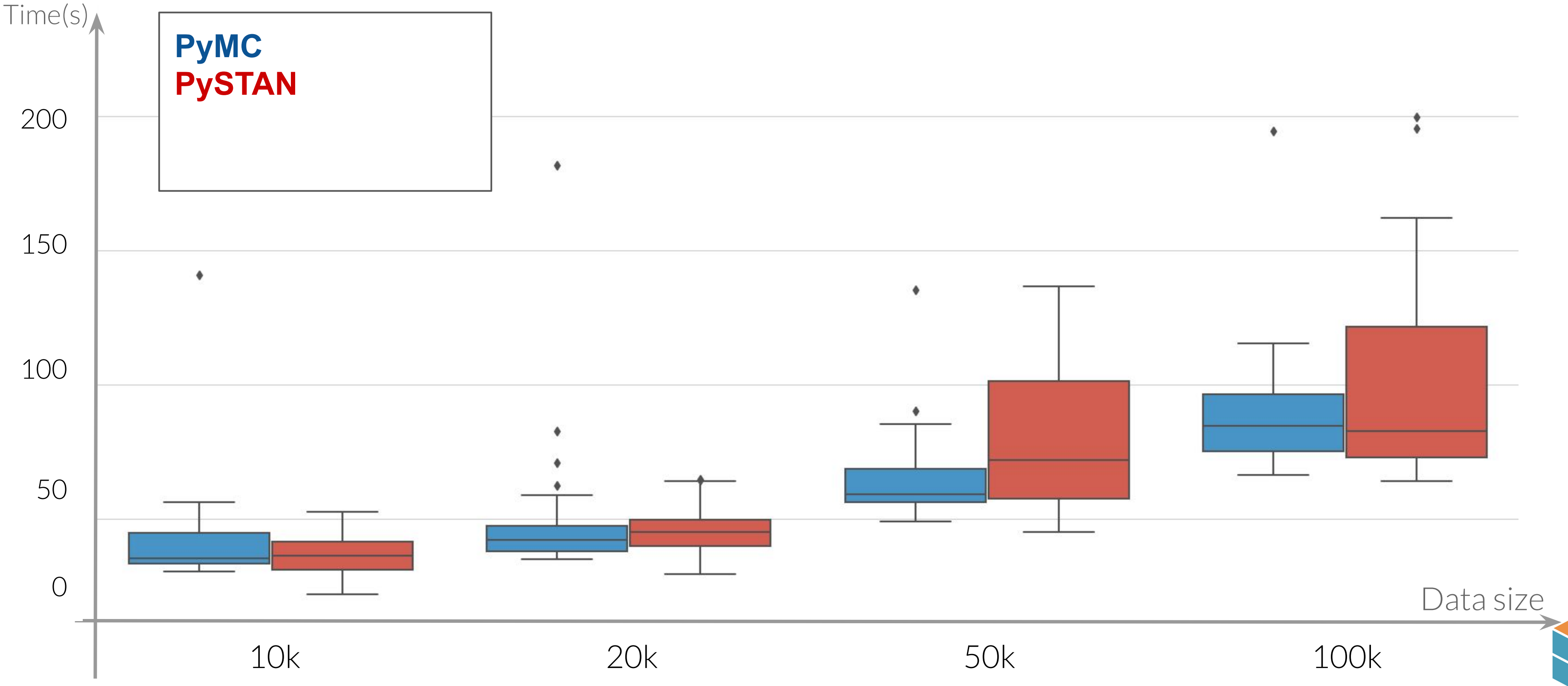
Comparison between PySTAN and PyMC

Using the Lifetime Value dataset for the tests, PySTAN outperforms PyMC for ‘small’ datasets and low-cardinality categories when using the **default samplers.** This is coherent with a benchmark conducted by Joshua Cook comparing STAN and PyMC3*

- As the data becomes larger (>20k) PyMC starts to performs better than PySTAN

* Joshua Cook’s benchmark compares PySTAN vs PyMC3, which is an older version of PyMC to what we have today and less optimized

Comparison between PySTAN and PyMC



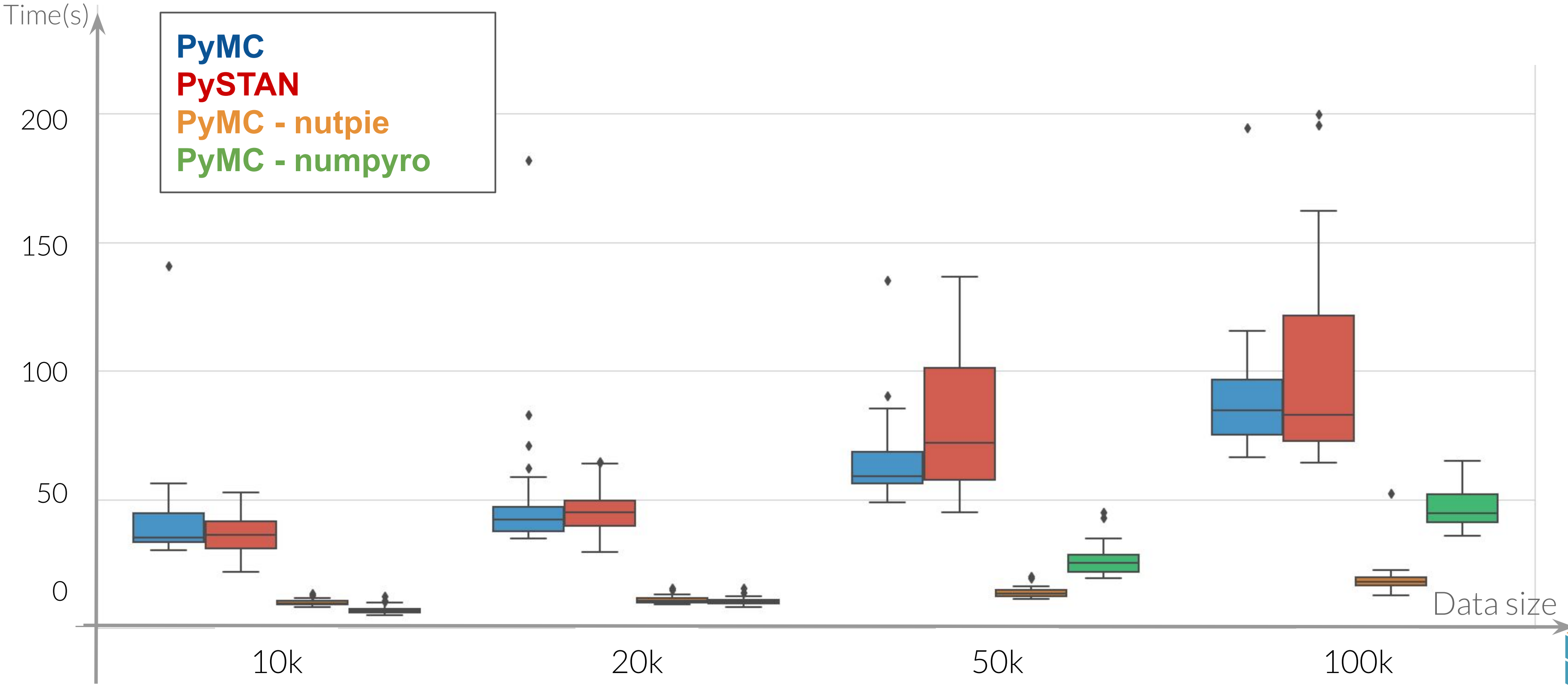
Comparison between PySTAN and PyMC

Using the Lifetime Value dataset for the tests, PySTAN outperforms PyMC for ‘small’ datasets and low-cardinality categories when using the **default samplers.** This is coherent with a benchmark conducted by Joshua Cook comparing STAN and PyMC3*

- As the data becomes larger (>20k) PyMC starts to performs better than PySTAN
- If using ‘*numpyro*’ or ‘*nutpie*’ samplers from PyMC, performance jumps up to 3x

*Joshua Cook’s benchmark compares PySTAN vs PyMC3, which is an older version of PyMC to what we have today and less optimized

Comparison between PySTAN and PyMC



Comparison between PySTAN and PyMC

One point worth of attention is that PyMC easily allows for multiple implementation of the same model through different classes. However, some of this classes can display significantly lower performance.

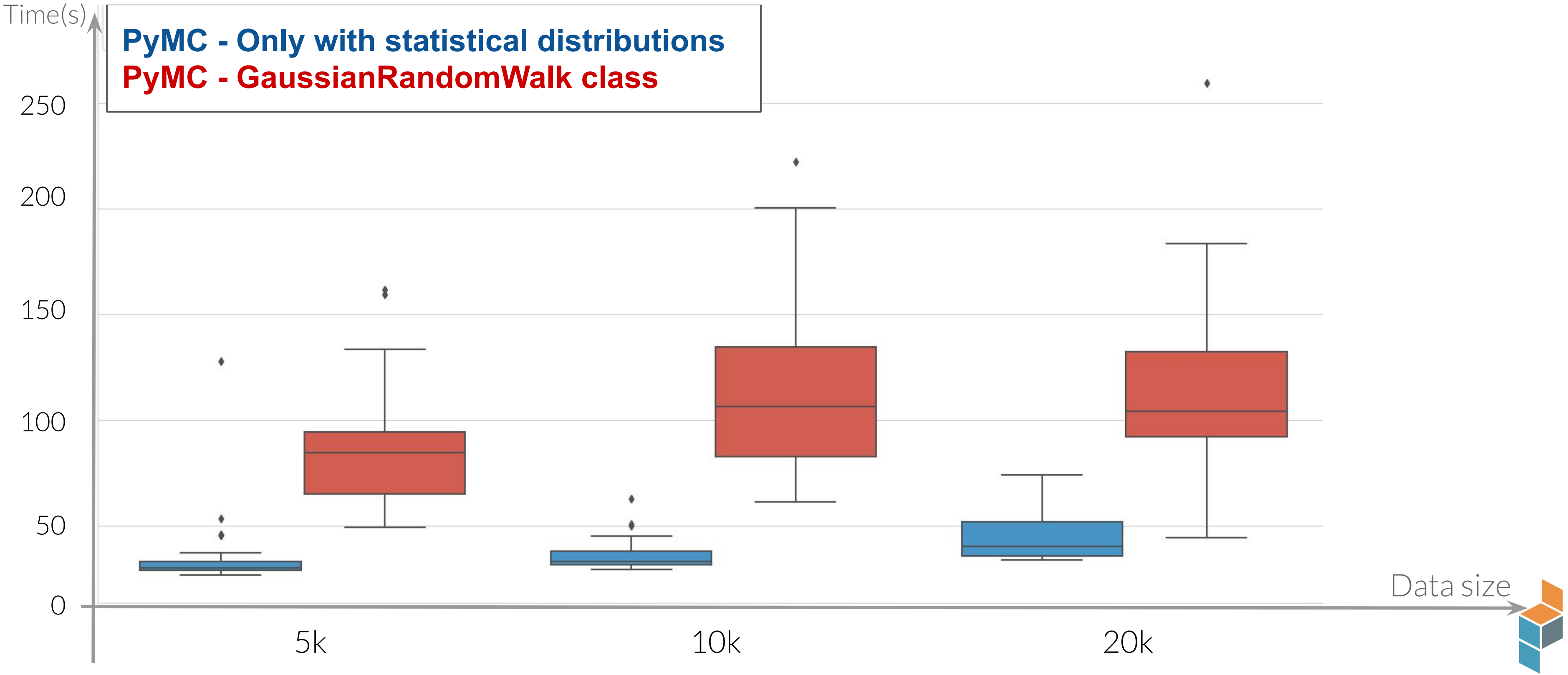
Comparison between PySTAN and PyMC

```
alpha = pm.Deterministic(
    "alpha", pt.concatenate([alpha_prior,
    pm.Normal("alpha_raw", sigma=alpha_dev, shape=n_days-1)]).cumsum(axis=0)
    )
```

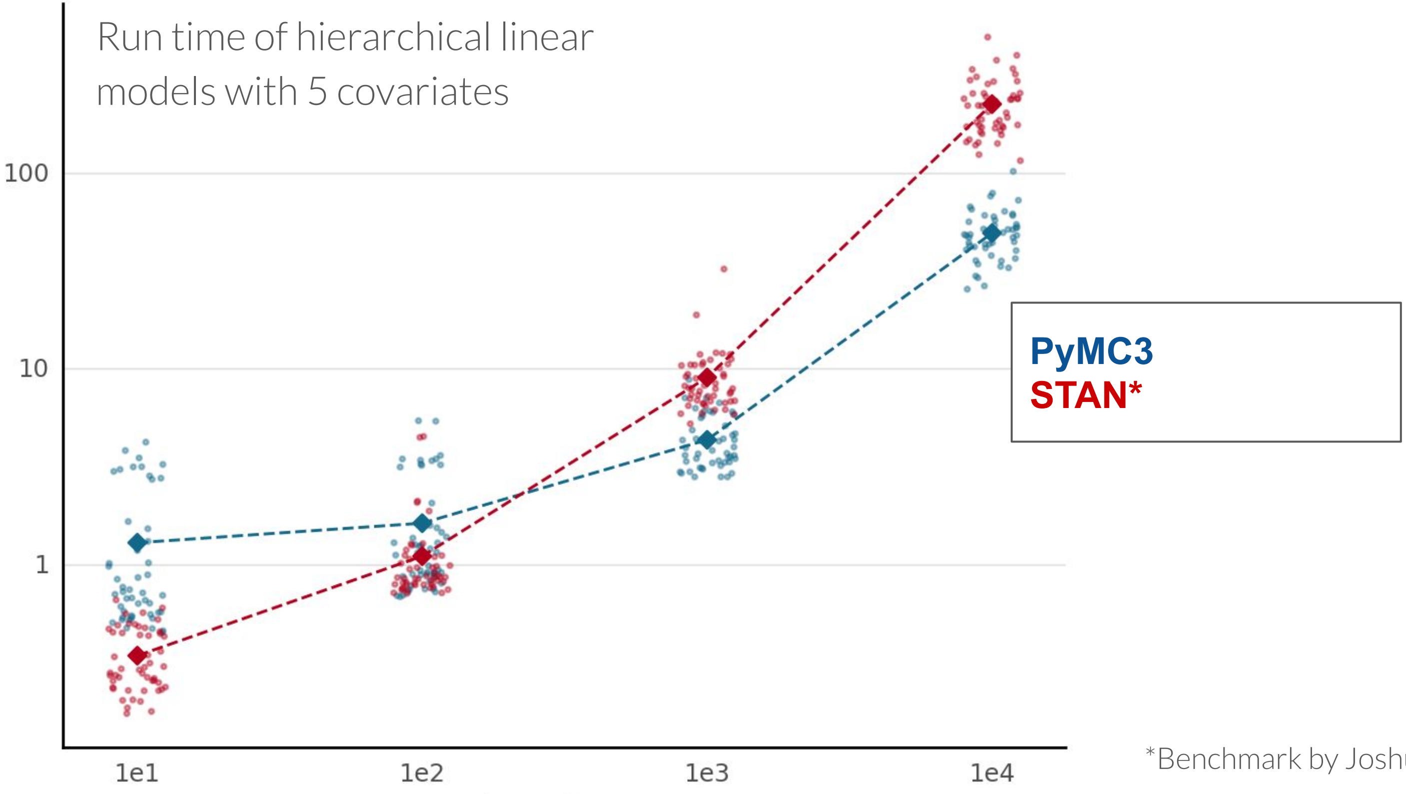
VS

```
alpha = pm.GaussianRandomWalk(
    "alpha", mu=0, sigma=alpha_dev, init_dist=pm.Normal.dist(6, 3), shape=n_days
    )
```

Comparison between PySTAN and PyMC



Comparison between PySTAN and PyMC(3)



*Benchmark by Joshua Cook

Comparison between PySTAN and PyMC

In summary

- Use PyMC if
    - Your dataset is large (>20K data points),
    - Models are complex (e.g. hierarchical models, high cardinality, many covariates),
    - There is a ready to use implementation available (e.g. PyMC Marketing Mix Models)
    - You can use in-development samplers numpyro or nutpie (require extra installations)

- Use PySTAN if
    - Your dataset isn't too large
    - Your model isn't too complex
    - You or colleges have experience working with STAN from another programming library (e.g. R)

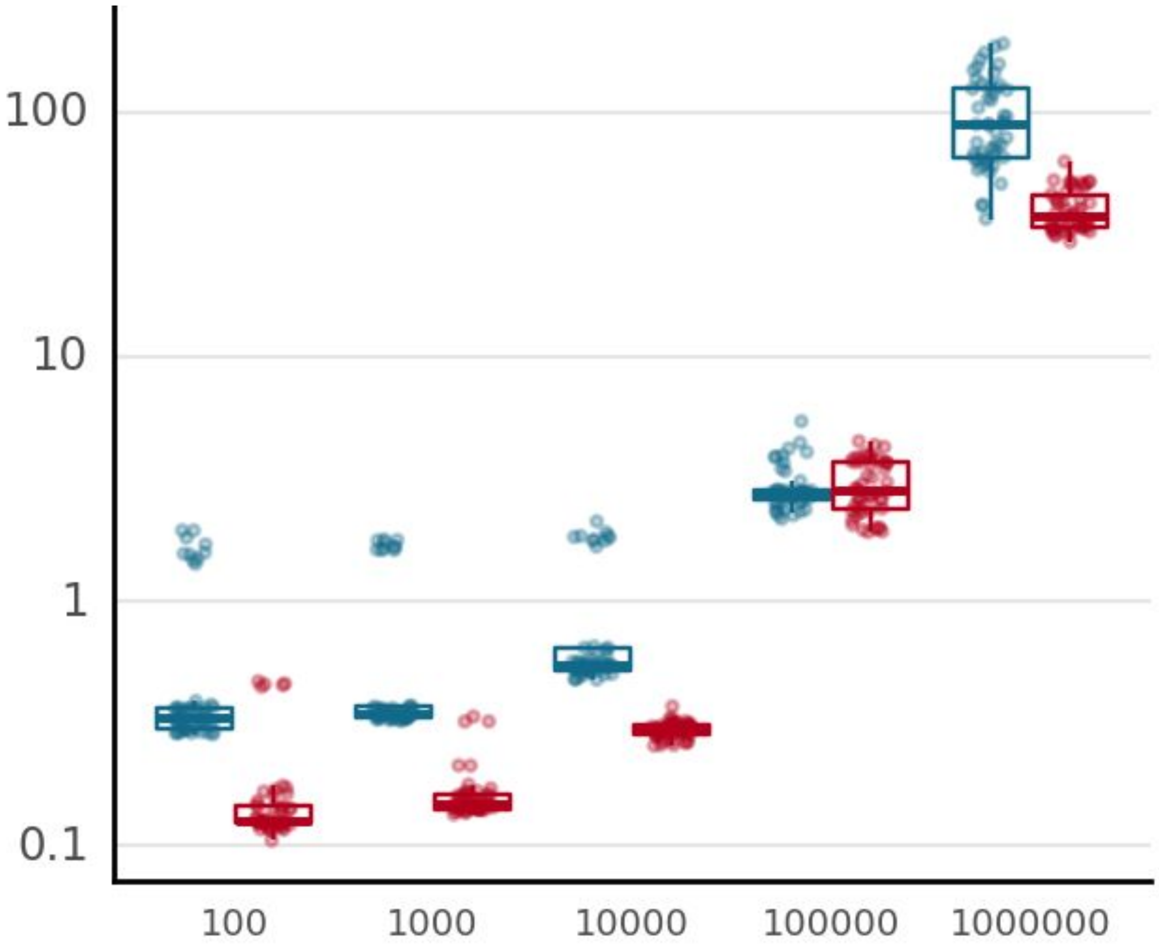
Conclusions

- Depending on the properties of your marketing campaigns, the uncertainty on your estimates change the strategy in operating them
- We saw how we can use PyMC or PySTAN to forecast LTV
- PyMC performs better than PySTAN for larger datasets or more complex models
- PyMC can have a steeper learning curve, but can be worthwhile if time is not critical

Special thanks

Q & A

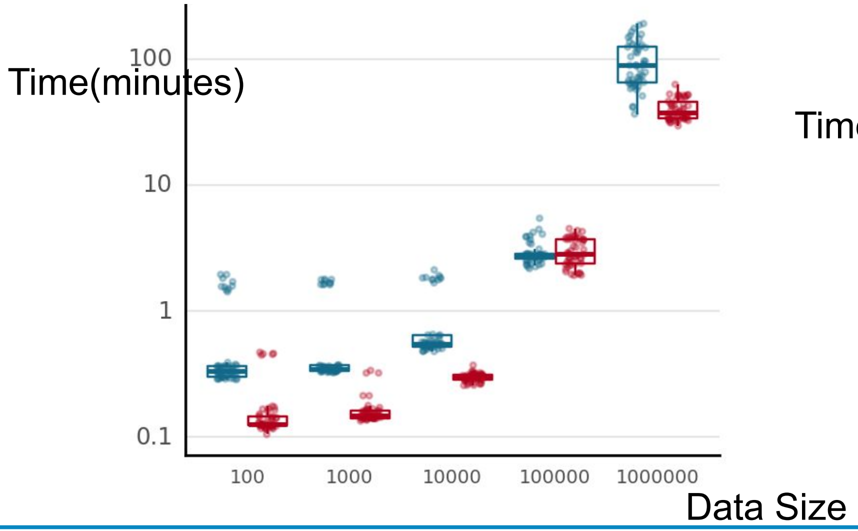
PyMC Faster Sampling with JAX and Numba

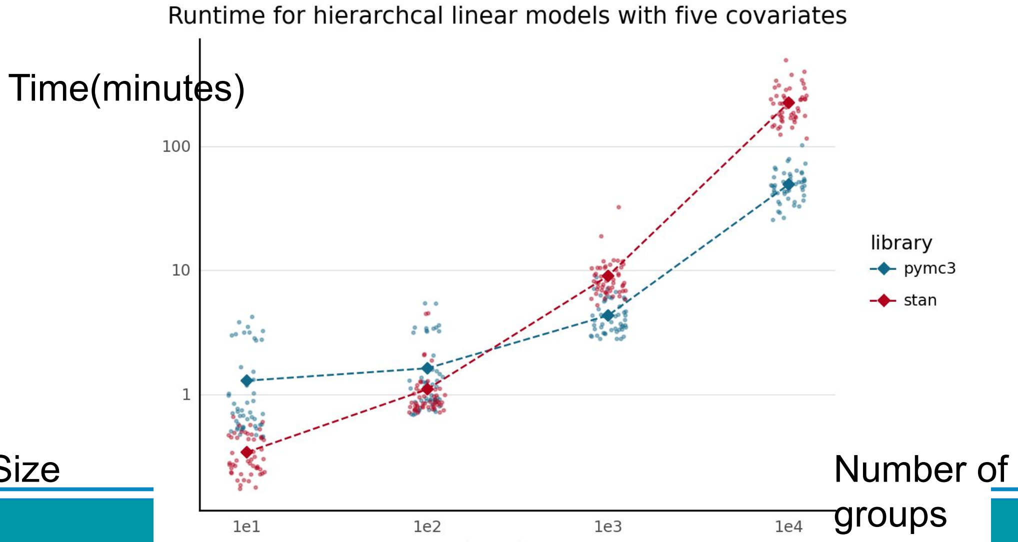


Number of groups

Comparison between PySTAN and PyMC - Speed

From my experience, and from benchmark conducted by Joshua Cook, PySTAN still outperforms PyMC for ‘small’ datasets and low-cardinality categories





https://github.com/pymc-devs/pymc-examples/blob/23e42c055c8203f1a57dd5726033b6068d02718b/examples/samplers/fast_sampling_with_jax_and_numba.ipynb

Comparison between PySTAN and PyMC - Resources

A perfect comparison on the number and quality of available resources on PySTAN and PyMC is never going to be perfect, but we can use some reference points:

- Average daily downloads (PyPiStats):
  - PySTAN 57.8k
  - PyMC: 19.2k
- Github stars
  - PySTAN: 0.3k
  - PyMC: 7.7k
- Ready to use models
  - PySTAN: Only statistical distributions
  - PyMC: many ready to use models (e.g. LTV, Medix Mix Modelling)

The (Py)STAN Model - Adapting the data for PySTAN

```
stan_format_data = {}
stan_format_data['N'] = int(data.shape[0])
stan_format_data['n_dates'] = len(data['join_date'].unique())
stan_format_data['date'] = list(data['join_date'].values)
stan_format_data['observation'] = list(data['target'].values)
```

PySTAN receives a data only as a dictionary

The PyMC Model

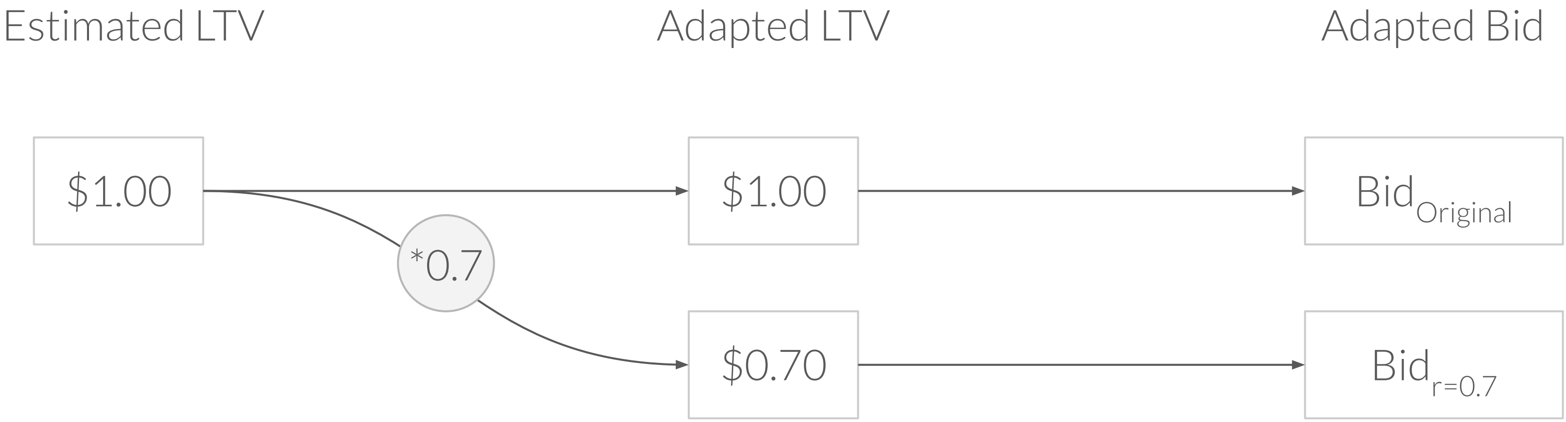
```
dates_tensor = pytensor.shared(data['join_date'].values)
target_tensor = pytensor.shared(data['target'].values)
n_days = len(data['join_date'].unique())
```

Similarly, PyMC receives data as ‘independent’ vectors

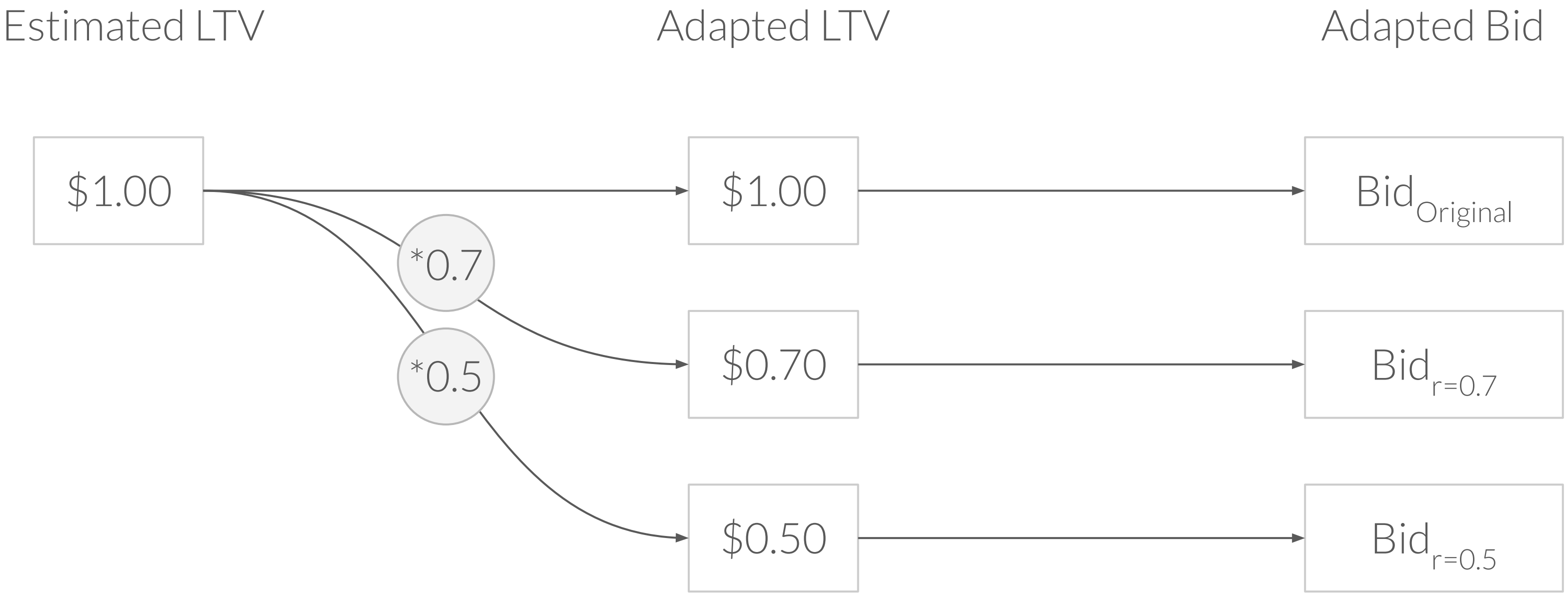
How to adapt the bidding strategy depending on the uncertainty around LTV

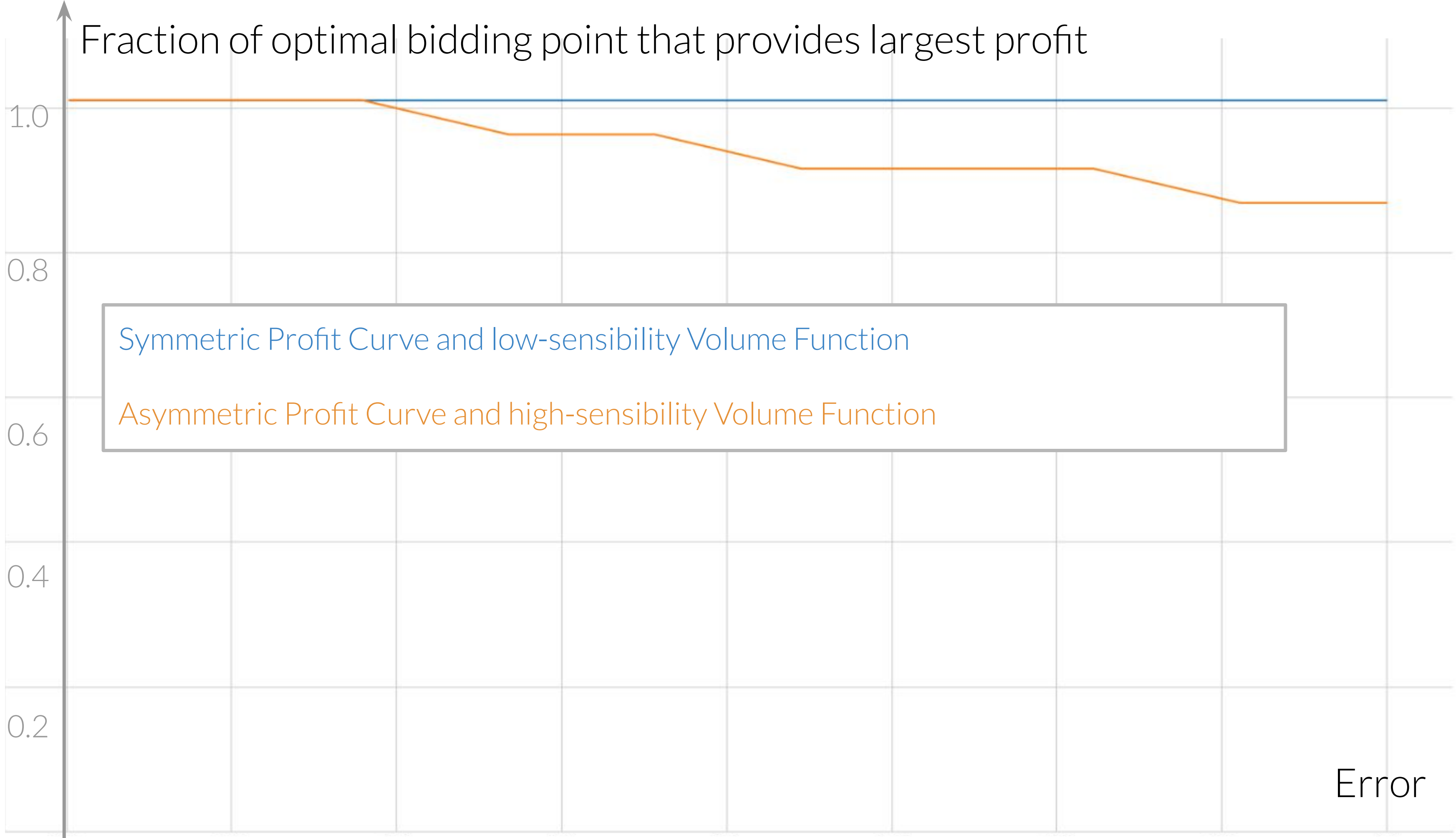
Estimated LTV		Adapted LTV		Adapted Bid
$1.00	→	$1.00	→	$Bid_{Original}$

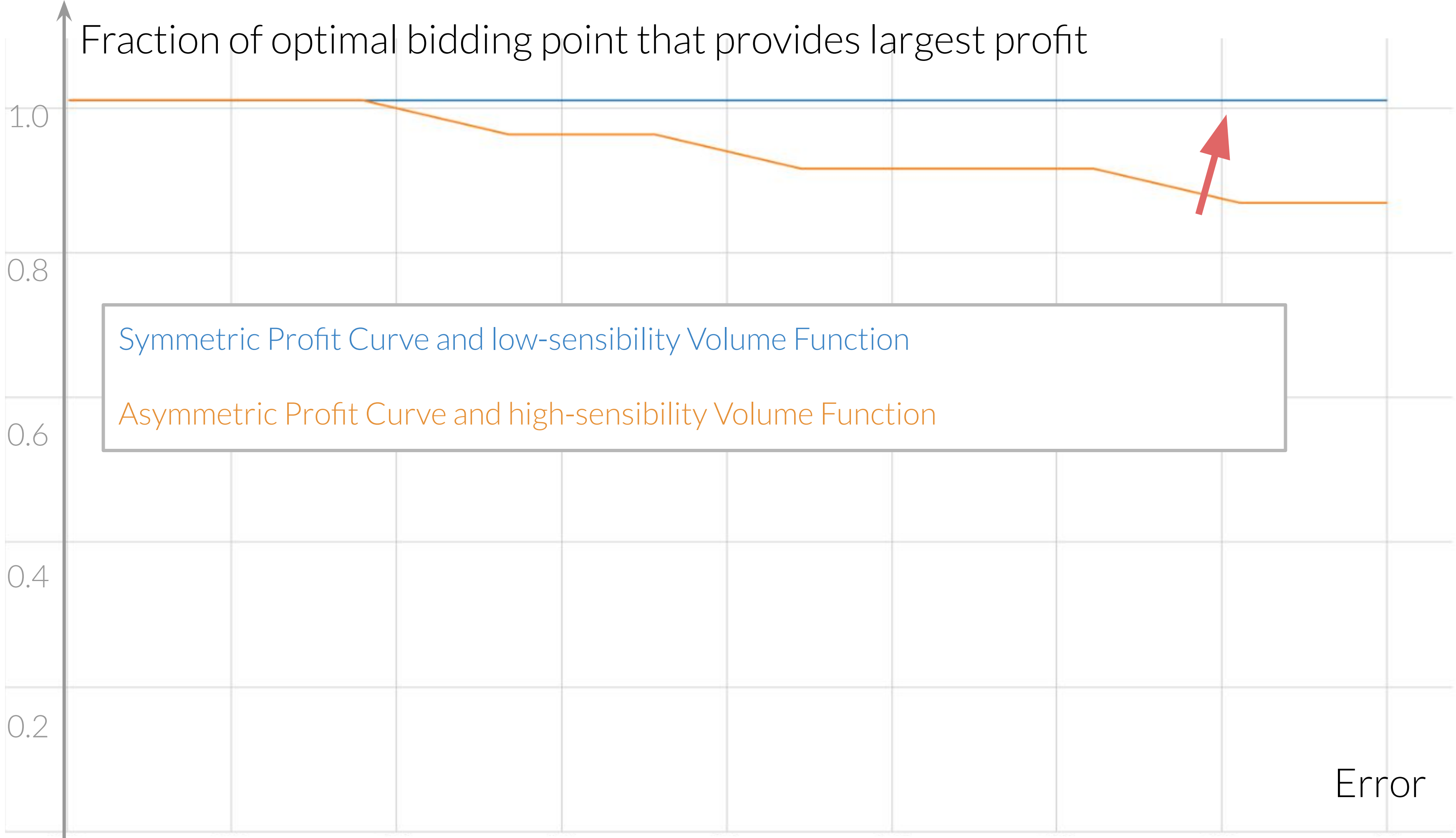
How to adapt the bidding strategy depending on the uncertainty around LTV

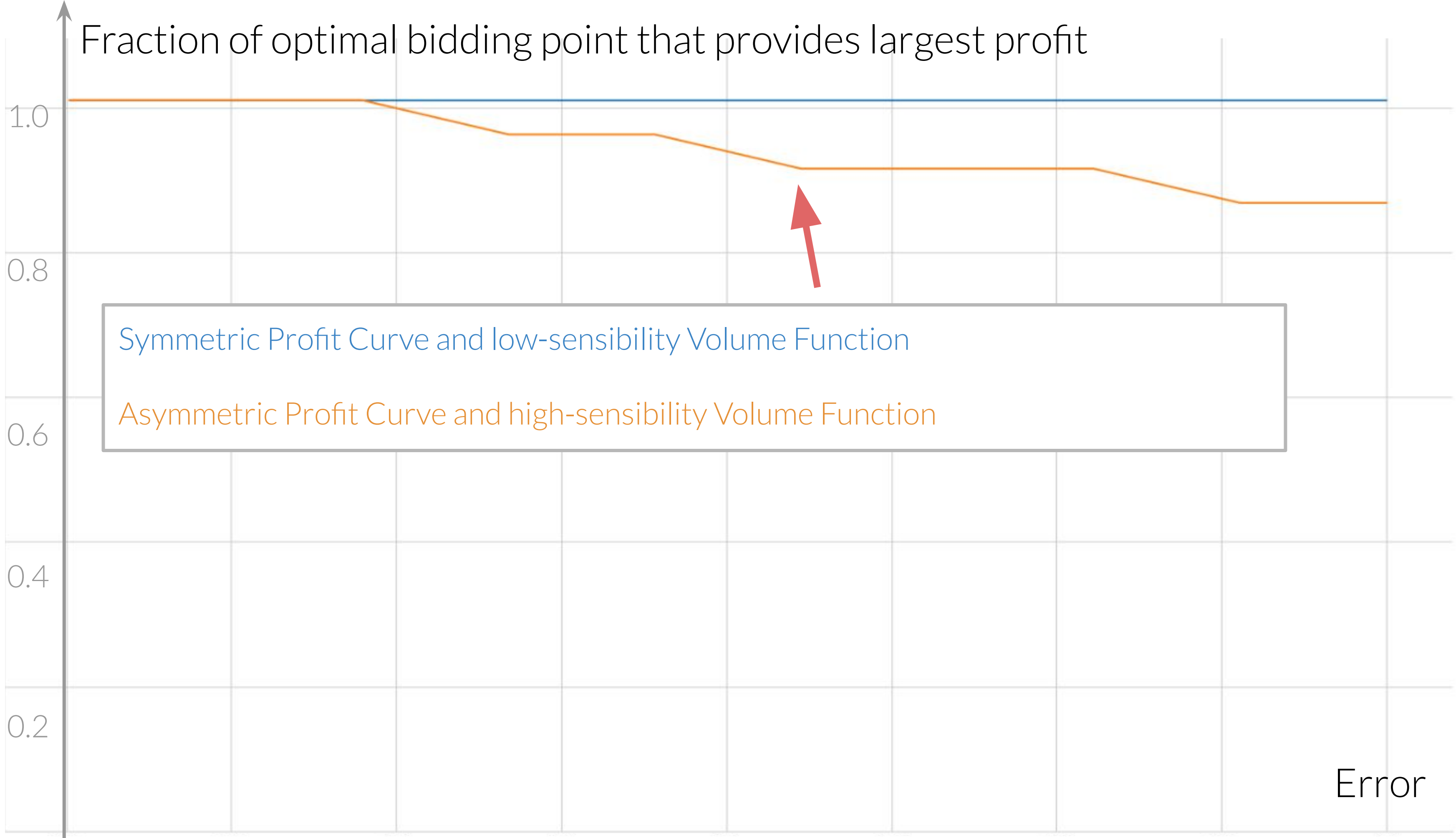


How to adapt the bidding strategy depending on the uncertainty around LTV




Fraction of optimal bidding point that provides largest profit
1.0
0.8
0.6
0.4
0.2
Symmetric Profit Curve and low-sensibility Volume Function
Asymmetric Profit Curve and high-sensibility Volume Function
Error


Fraction of optimal bidding point that provides largest profit
1.0
0.8
0.6
0.4
0.2
Symmetric Profit Curve and low-sensibility Volume Function
Asymmetric Profit Curve and high-sensibility Volume Function
Error


Fraction of optimal bidding point that provides largest profit
1.0
0.8
0.6
0.4
0.2
Symmetric Profit Curve and low-sensibility Volume Function
Asymmetric Profit Curve and high-sensibility Volume Function
Error