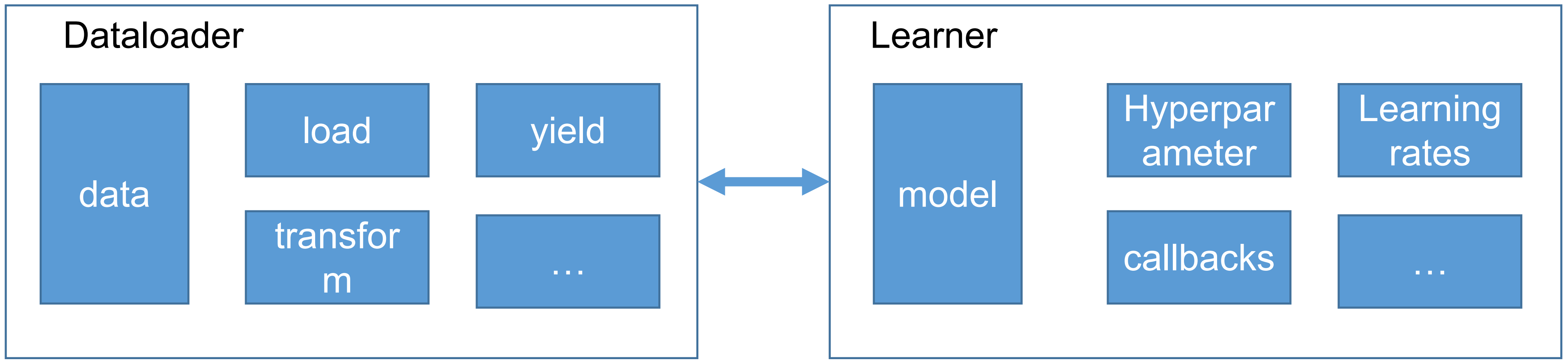
tabint

ML workflow

- At very basic, ML workflow contains ML entities and theirs relationship
- At programming perspective: We have class of entities and class that handle theirs relationship



Deep learning workflow

Raw dataset: in file or memory

Dataset class:
contain raw data for trn/val/tst
or method to read, transform data

Data loader class:
use method to read and provide data when learner request

Model class: model architechture

Learner class: manage all things outsite model like hyper parameter, callbacks, learning rates….
Request dataloader to provide data

Non-Deep learning workflow

Dataset class:
Load all file in one time to memory

Data loader class:
Don’t need because data is provided one time to model

Model class: model architechture – all is specify by parameter of model

Learner class: Not too importance. Base on model parameter. All is compress to a single class.

➡ In DL we need to handle all part in process. In non-DL, nearly all the things can be done by a specific library

What tabint focus on

- Flexible and convenient
- Tree base method:
    - Decision tree
    - Random tree (random forest)
    - Boosting tree
- Avoid common pitfall
- Interpretation, explaination
- Try to do things in parallel (dask, numba…) – in future

Flexible and convenient

Object-oriented programming

@classmethod
As a input

Class

init
Variable or method

Function as a method to gerenate output

Function as a method to change class init

@property as a property of class

Flexible

Avoid to create a nested class

<u>Traditional approach</u>

Class Dataset
Class Dataloader()
        init: Dataset

Class Model
Class Learner
        init: Dataloader, Model

So we have a big class Learner by nested dataset, dataloader and model

<u>tabint appoarch</u>

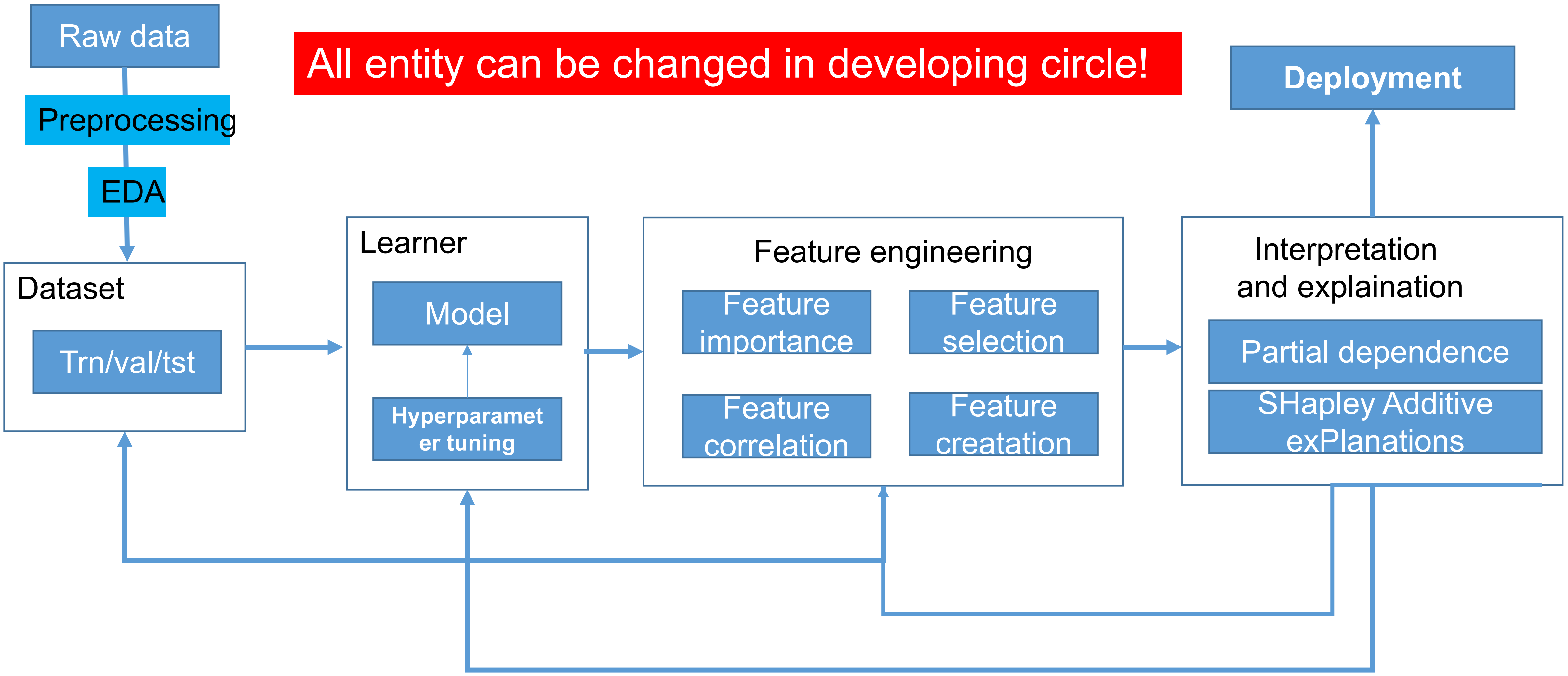
Class Dataset

Class Model

Class Learner
        input: Dataset, Model

Learner's input is dataset and model

Why we should use this approach



Convenient

If a method is used many time again and again just define it as a function of related entity (class)

```python
class TBDataset:
    def __init__(self, x_trn, y_trn, x_val, y_val, x_tst = None):
        self.x_trn, self.y_trn = x_trn, y_trn
        self.x_val, self.y_val = x_val, y_val
        self.x_tst = x_tst

    @classmethod
    def from_SklearnSplit(cls, df, y_df, ratio = 0.2, x_tst = None, **kargs):
        x_trn, x_val, y_trn, y_val = train_test_split(df, y_df, test_size=ratio, stratify = y_df)
        return cls(x_trn, y_trn, x_val, y_val, x_tst)

    def val_permutation(self, cols):
        cols = to_list(cols)
        df = self.x_val.copy()
        for col in cols: df[col] = np.random.permutation(df[col])
        return df

    def add(self, col, f, inplace = True, tp = 'trn'):
        if inplace:
            for df in [self.x_trn, self.x_val, self.x_tst]: if df is not None: df[col] = f(df)
        else:
            if tp == 'tst':
                df = self.x_tst.copy()
                df[col] = f(df)
                return df
            else:
                df, y_df = self.x_trn[col], self.y_trn if tp == 'trn' else self.x_trn[col], self.y_trn
                df[col] = f(df)
                return df, y_df
```

```python
    def sample(self, tp = 'trn', ratio = 0.3):
        if 'tst' == tp:
            return None if self.x_tst is None else self.x_tst.sample(self.x_tst.shape[0]*ratio)
        else:
            df, y_df = self.x_trn[col], self.y_trn if tp == 'trn' else self.x_trn[col], self.y_trn
            _, df, _, y_df = train_test_split(df, y_df, test_size = ratio, stratify = y_df)
            return df, y_df

    def keep(self, col, inplace = True, tp = 'trn'):
        if inplace:
            for df in [self.x_trn, self.x_val, self.x_tst]: if df is not None: df = df[col]
        else:
            return {'trn': self.x_trn[col], self.y_trn, 'val': self.x_val[col], self.y_val, 'tst': self.x_tst[col]}[tp]

    def drop(self, col, inplace = True, tp = 'trn'):
        if inplace:
            for df in [self.x_trn, self.x_val, self.x_tst]: if df is not None: df.drop(col, axis=1, inplace = True)
        else:
            return {'trn': self.x_trn.drop(col, axis = 1), self.y_trn,
                    'val': self.x_val.drop(col, axis = 1), self.y_val,
                    'tst': self.x_tst.drop(col, axis = 1)}[tp]

    def trn_n_val(self): return self.x_trn, self.y_trn, self.x_val, self.y_val

    @property
    def features(self): return self.x_trn.columns
```

Focus on Tree base method

With excellent performance on all eight metrics, calibrated boosted trees were the best learning algorithm overall. Random forests are close second.
Caruana et al. made in [empirical comparison of supervised learning algorithms]

One hot encoding in pre-processing step -> Better for split with categorical variable

```
for n,c in df.items(): numericalize(df, c, n, max_n_cat)
```

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Focus on Tree base method

Learner for lighgbm

Randomforest
and xgboost is not done, yet!

```python
class LGBLearner:
    def __init__(self, fn = 'LGB_Model.pkl'):
        self.fn = fn
        self.score = []

    def fit(self, params, x_trn, y_trn, x_val, y_val, ctn = False, save = True, early_sto
        self.md = None
        if ctn: self.load()
        lgb_trn, lgb_val = self.build_ds(x_trn, y_trn, x_val, y_val)
        self.md = lgb.train(params = params,
                            train_set = lgb_trn,
                            valid_sets = [lgb_trn, lgb_val],
                            init_model = self.md,
                            early_stopping_rounds = early_stopping_rounds,
                            verbose_eval = verbose_eval, **kargs)

        self.score.append(self.md.best_score)
        if save: self.save()

    @staticmethod
    def build_ds(x_trn, y_trn, x_val, y_val):
        lgb_trn = lgb.Dataset(x_trn, y_trn)
        lgb_val = lgb.Dataset(x_val, y_val, free_raw_data=False, reference=lgb_trn)
        return lgb_trn, lgb_val

    def predict(self, df, **kargs): return self.md.predict(df, **kargs)

    def load(self):
        with open(self.fn, 'rb') as fin: self.md = pickle.load(fin)

    def save(self, fn = None):
        fn = isNone(fn, self.fn)
        with open(fn, 'wb') as fout: pickle.dump(self.md, fout)
```

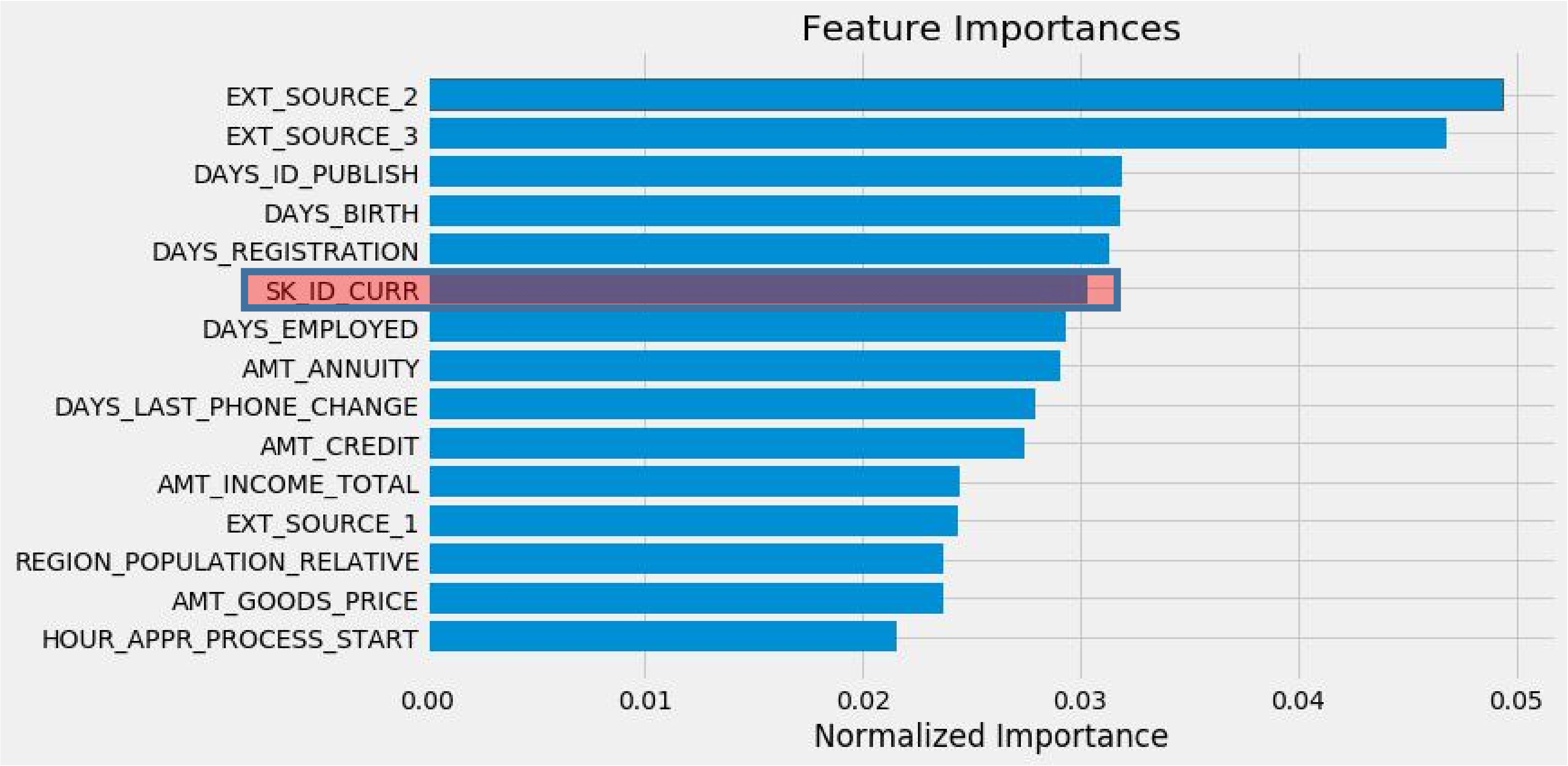
Focus on Tree base method

Interpretation and explaination for Tree base method

```python
class SHAP:
    def __init__(self, explainer, shap_values, df, features):
        shap.initjs()
        self.explainer = explainer
        self.shap_values = shap_values
        self.df, self.features = df, features

    @classmethod
    def from_Tree(cls, learner, ds, sample = 10000):
        df = ds.x_trn.sample(sample).astype(np.float32)
        explainer = shap.TreeExplainer(learner.md)
        shap_values = explainer.shap_values(df)
        features = df.columns
        return cls(explainer, shap_values, df, features)
```

Avoid common pitfall

Feature importance



https://github.com/WillKoehrsen/kaggle-credit-data-science-competition/blob/master/notebooks/1-Getting%20Started.ipynb

Avoid common pitfall

Feature importance

Feature importance method can be bias and wrong
-> Solution: permutation importance
See more at: https://medium.com/@kien.vu/8d60ed8ce314

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24

Avoid common pitfall

permutation importance. How it implemented in tabint

Method in dataset

```
def val_permutation(self, cols):
    cols = to_list(cols)
    df = self.x_val.copy()
    for col in cols: df[col] = np.random.permutation(df[col])
    return df
```

Don’t use nested class. Pls!

Importance class

```
class Importance:
    def __init__(self, impt_df):
        self.I = sort_desc(impt_df)

    @classmethod
    def from_Learner(cls, learner, ds,  group_cols, score = roc_auc_score):
        '''
        http://explained.ai/rf-importance/index.html
        '''
        y_pred = learner.predict(ds.x_val)
        baseline = score(ds.y_val, y_pred)
        I = pd.DataFrame.from_dict({'Feature': [' & '.join(to_list(cols)) for cols in group_cols]})
        I['Importance'] = I.apply(cls.cal_impt, axis = 1, learner = learner, ds = ds, baseline = baseline, score = score)
        return cls(I)

    @staticmethod
    def cal_impt(x, learner, ds, baseline, score):
        cols = x[0].split(' & ')
        y_pred_permut = learner.predict(ds.val_permutation(cols))
        permut_score = score(ds.y_val, y_pred_permut)
        return baseline - permut_score

    def top(self, n): return [col.split(' & ') for col in self.I.Feature[:n]]

    def plot(self, **kagrs): plot_barh(self.I, **kagrs)
```

Avoid common pitfall

Split train/valid/test set. What is problem?

Model need to be good in both data it seen and not seen.

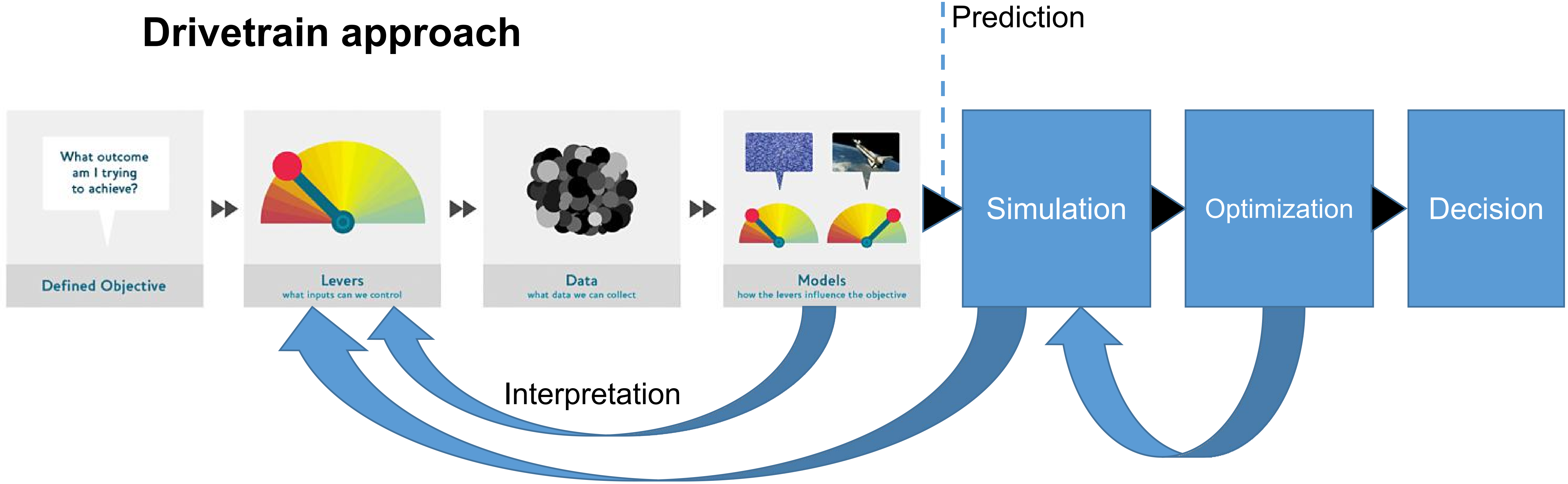
-> valid set need similar to test set and contain data that not in training set. See more at: https://medium.com/@kien.vu/d6b7a8dbaaf5

Avoid common pitfall

Split train/valid/test set.
How it implemented in tabint.
It is a input method of dataset!

```
class TBDataset:
    def __init__(self, x_trn, y_trn, x_val, y_val, x_tst = None):
        self.x_trn, self.y_trn = x_trn, y_trn
        self.x_val, self.y_val = x_val, y_val
        self.x_tst = x_tst

    @classmethod
    def from_SklearnSplit(cls, df, y_df, ratio = 0.2, x_tst = None, **kargs):
        x_trn, x_val, y_trn, y_val = train_test_split(df, y_df, test_size=ratio, stratify = y_df)
        return cls(x_trn, y_trn, x_val, y_val, x_tst)

    @classmethod
    def from_TBSplit(cls, df, y_df, x_tst, pct = 2, ratio = 0.2, **kargs):
        _, cats = get_cons_cats(df)

        tst_key = x_tst[cats].drop_duplicates().values
        tst_key = set('~'.join([str(j) for j in i]) for i in tst_key)

        df_key = df[cats].apply(lambda x: '~'.join([str(j) for j in x.values]), axis=1)
        mask = df_key.isin(tst_key)

        x_trn, y_trn = df[~mask], y_df[~mask]
        x_val_set, y_val_set = df[mask], y_df[mask]

        x_val = x_val_set.groupby(cats).apply(cls.random_choose, pct, ratio, **kargs)
        val_index = set([i[-1] for i in x_val.index.values])
        x_val.reset_index(drop=True, inplace=True)

        mask = x_val_set.index.isin(val_index)
        y_val = y_val_set[mask]
        x_trn, y_trn = pd.concat([x_trn, val_set[~mask]]), pd.concat([y_trn, y_val_set[~mask]])

        return cls(x_trn, y_trn, x_val, y_val, x_tst)
```

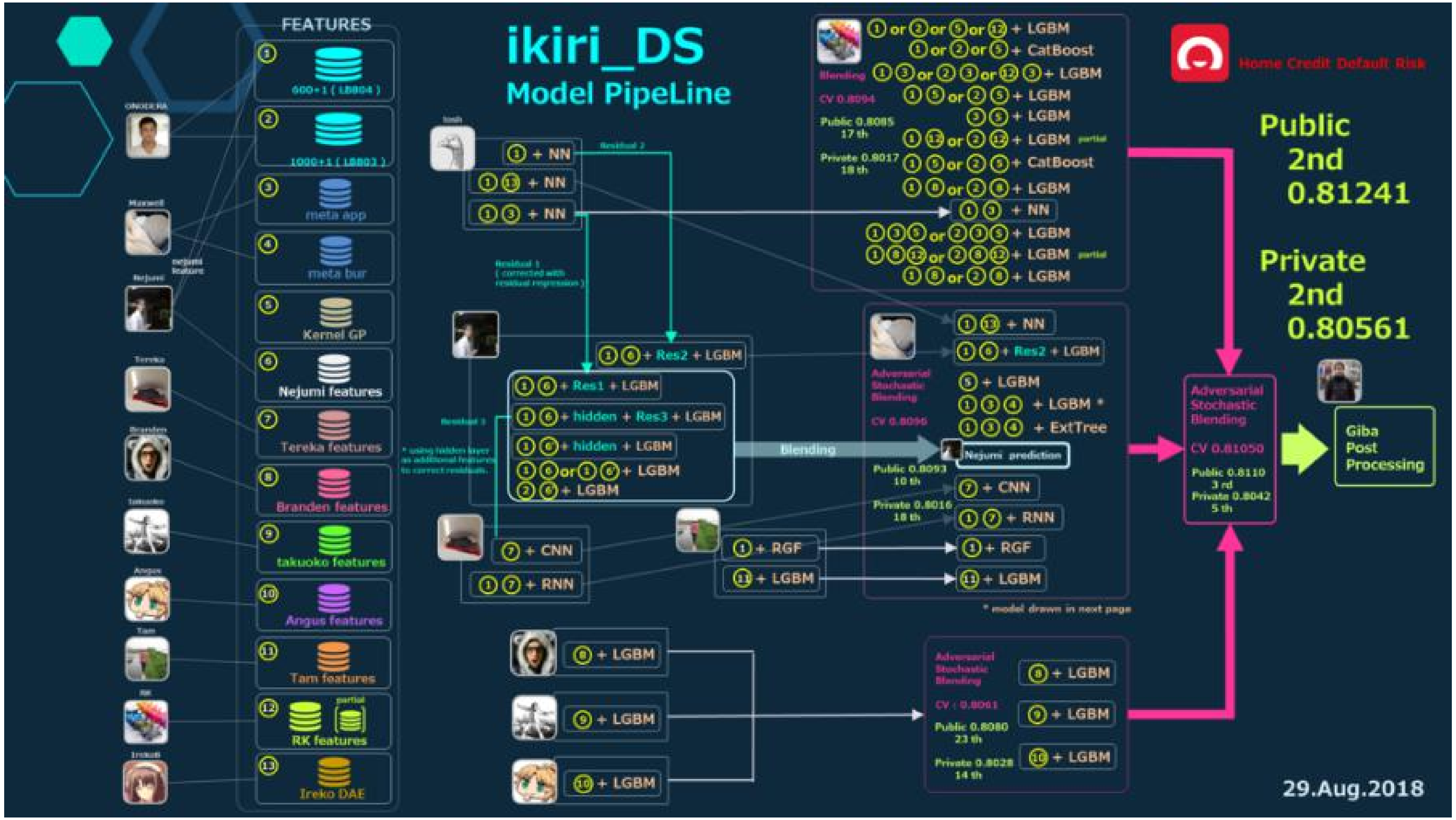
Focus on interpretation, explaination. Why?



https://www.oreilly.com/ideas/drivetrain-approach-data-products

Focus on interpretation, explaination

They gone so far but in the wrong way



https://www.kaggle.com/c/home-credit-default-risk/discussion/64722

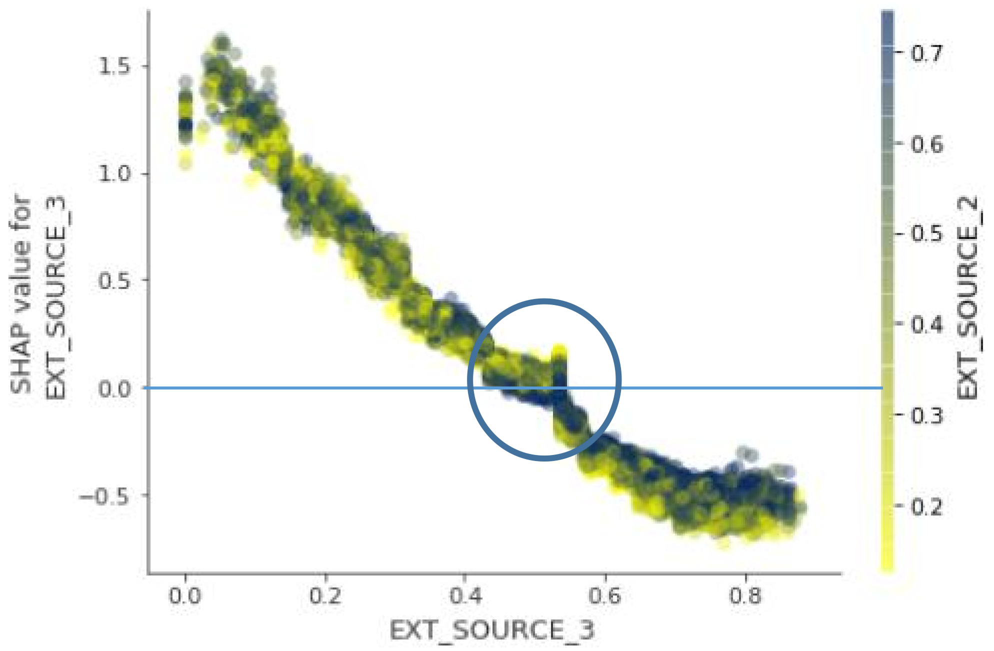
Focus on interpretation, explaination

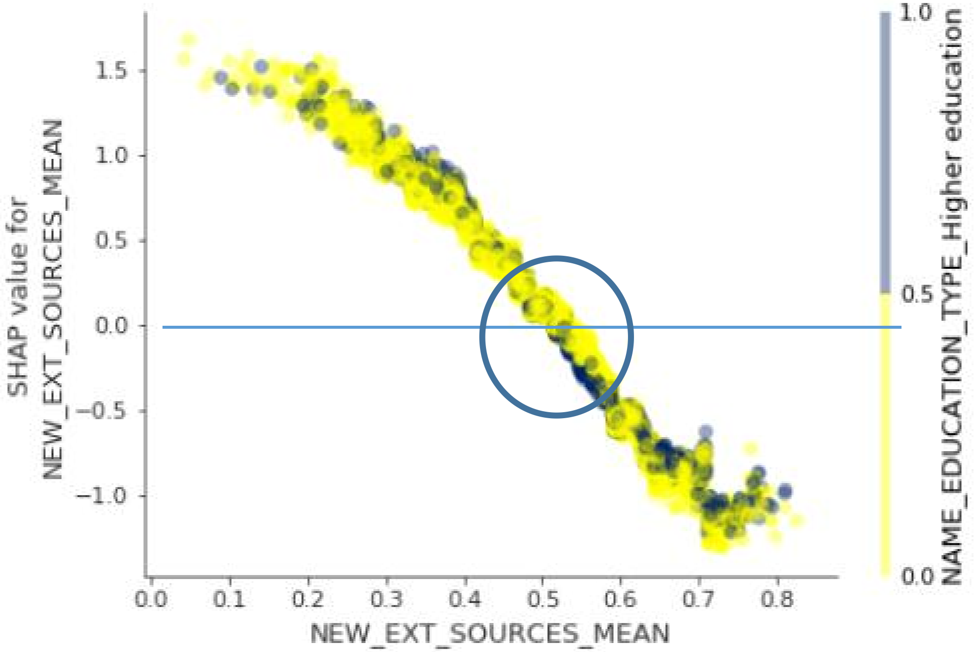
Understanding better. Doing and acting better!

Focus on interpretation, explaination

Understanding better. Doing and acting better

Feature engineering. Why it works?

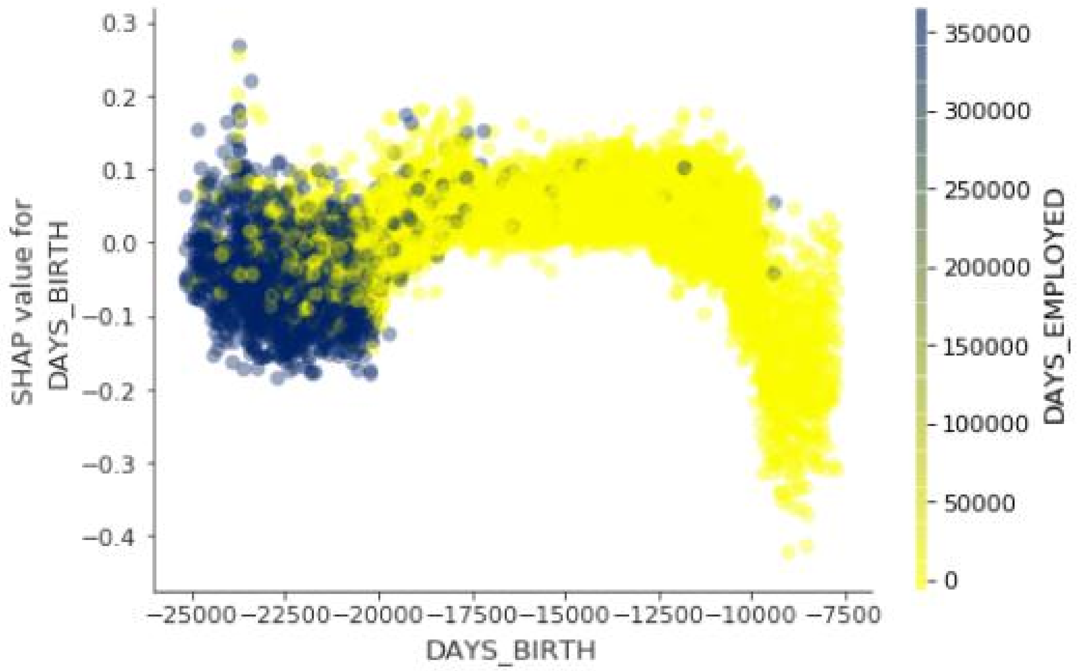


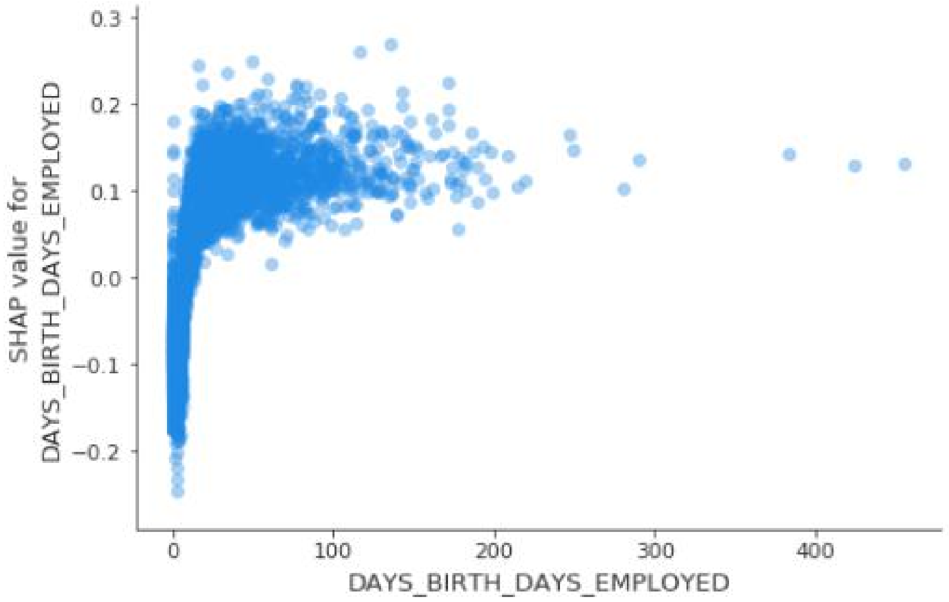


Focus on interpretation, explaination

Understanding better. Doing and acting better

Feature engineering. DAYS_BIRTH and DAYS_EMPLOYED.Why it works?



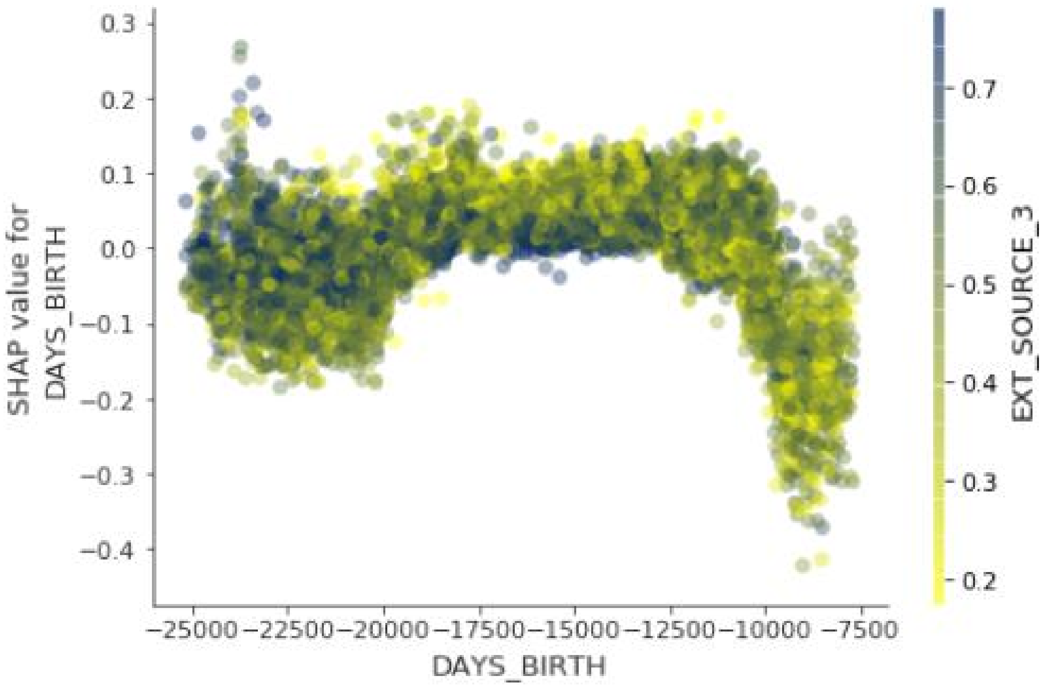


Focus on interpretation, explaination

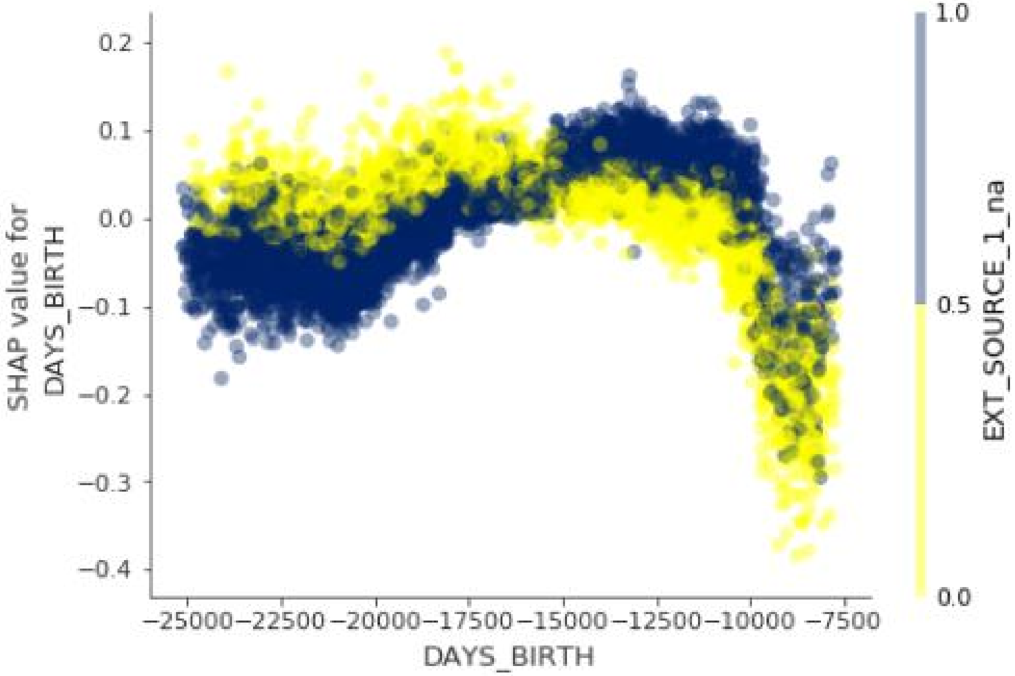
Understanding better. Doing and acting better

Feature engineering. DAYS_BIRTH and EXT_SOURCE_3.

Why it does not work?



What is better?



How to start

- Developing
- Debuging