Overview

1. Exploratory Data Analysis (EDA): what and why?
2. Things to explore
3. Exploration and visualization tools
4. (A bit of) dataset cleaning
5. Kaggle competition EDA

Motivating example

id	...	# promos sent	# promos used	*diff*	*used this promo?*
13	...	0	0	1	**1**
13	...	1	1	0	**0**
13	...	2	1	1	**0**
13	...	4	2	1	**1**
13	...	5	3	1	**1**
13	...	6	3	NaN	**0**

1. For each person sort by ‘**# promos sent**’
2. Look at difference between consecutive rows in ‘**# promos used**’ column (‘***diff***’ feature)

Check if the data is intuitive

...	*Age*	...
...	21	...
...	45	...
...	336	...
...	19	...
...	...	...

- Is *336* a typo?
- Or we misinterpret the feature and age 336 is normal

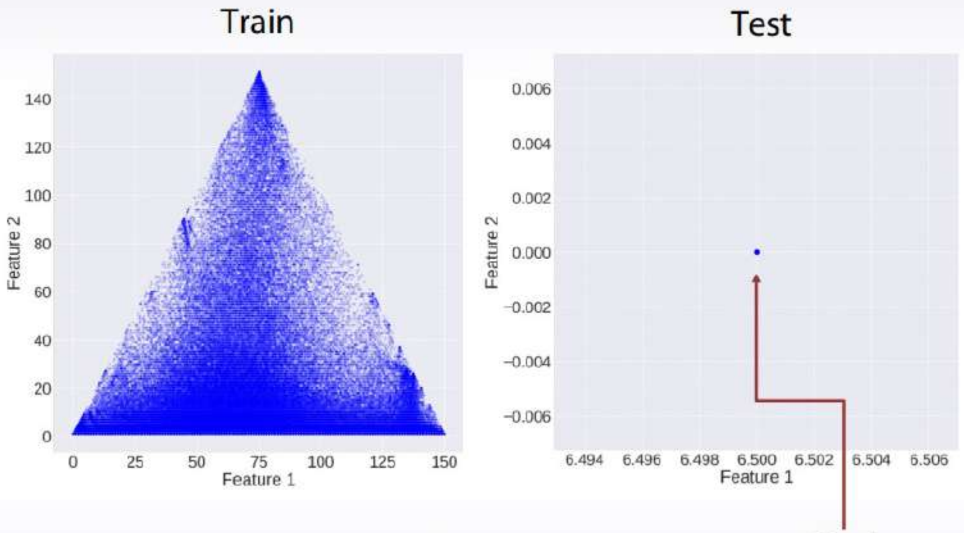
Check if the data is intuitive

Task: Predict advertiser’s cost

Data:

AdGroupId	AdNetwork Type2	MaxCpc	Slot	Clicks	Impressions	*is_incorrect*
78db044136	s	0.28	s_2	*3*	*0*	*True*
68a0110c33	s	1	s_2	1	13	*False*
2r39fw11w3	p	1.2	p_1	3	419	*False*

Understand how the data was generated



Dot!

#days in *train* > #days in *test*
#rows in *train* < #rows in *test*

Conclusion

- **Get domain knowledge**
  - It helps to deeper understand the problem
- **Check if the data is intuitive**
  - And agrees with domain knowledge
- **Understand how the data was generated**
  - As it is crucial to set up a proper validation

Anonymized data

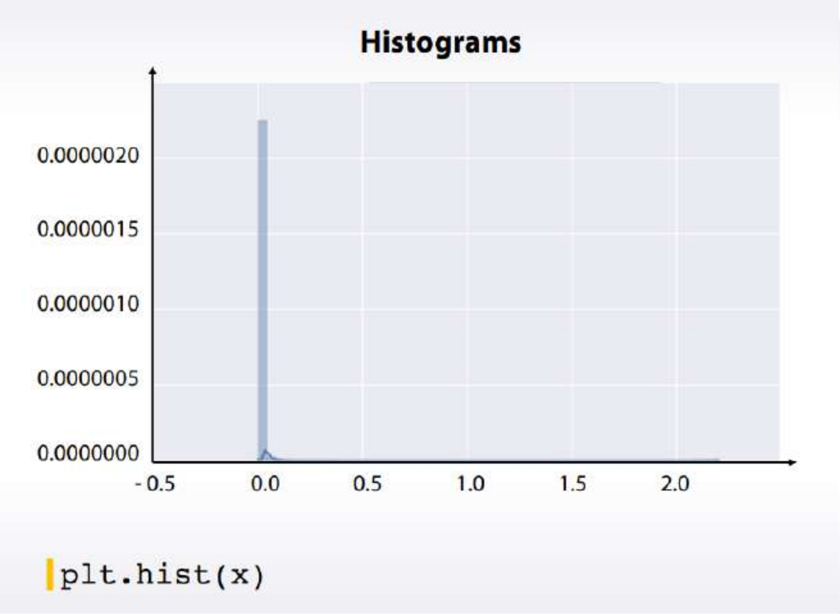
Text	Encoded text
I want this table	7ugy 972h 98ww hj34
Table is what I want	hj34 4f08 rtte 7ugy 972h
This table is red	98ww hj34 4f08 4rj9
And this is me	jk8r 98ww 4f08 9jo4

- Two things to do with anonymized features:
  - **Try to decode the features**
    - Guess the true meaning of the feature
  - **Guess the feature types**
    - Each type needs its own preprocessing

Anonymized data

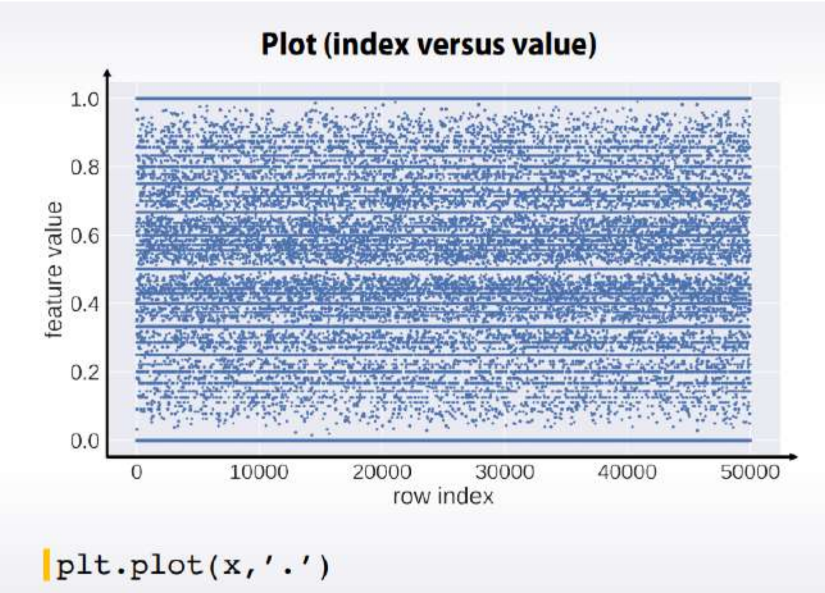
id	x1	x2	x3	x4	x5	x6
1	m268i97y	0	NO	105.4	14	
2	j0gheu6	1	YES	25.631	12	
3	26fmsp6u	1	NO	12.0	12	m268i97y
4	13e5dpzp	0	NO	140.12	14	m268i97y

- Explore individual features
  - Guess the meaning of the columns
  - Guess the types of the column
- Explore feature relations
  - Find relations between pairs
  - Find feature groups



Histograms

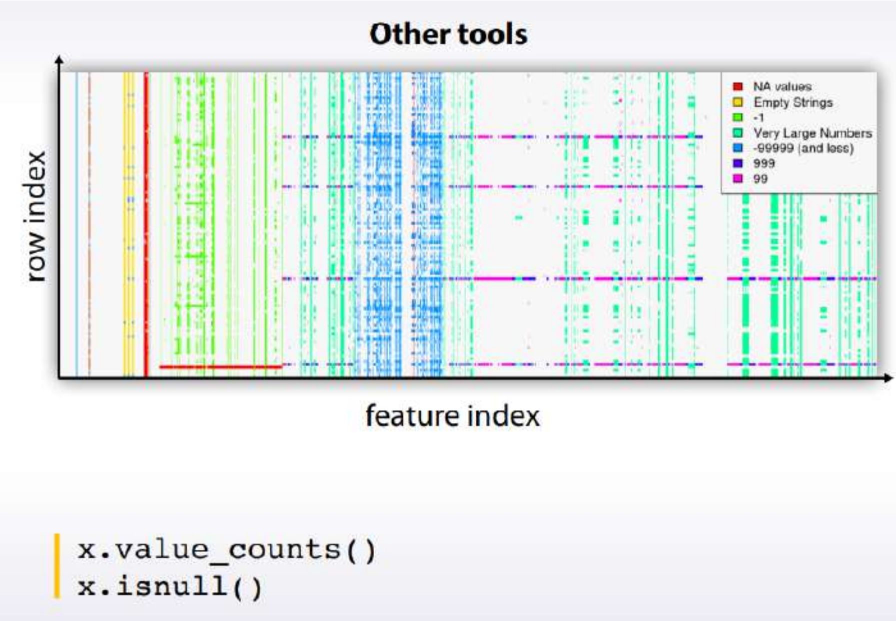
```
plt.hist(x)
```



Plot (index versus value)

```
plt.plot(x,'.')
```

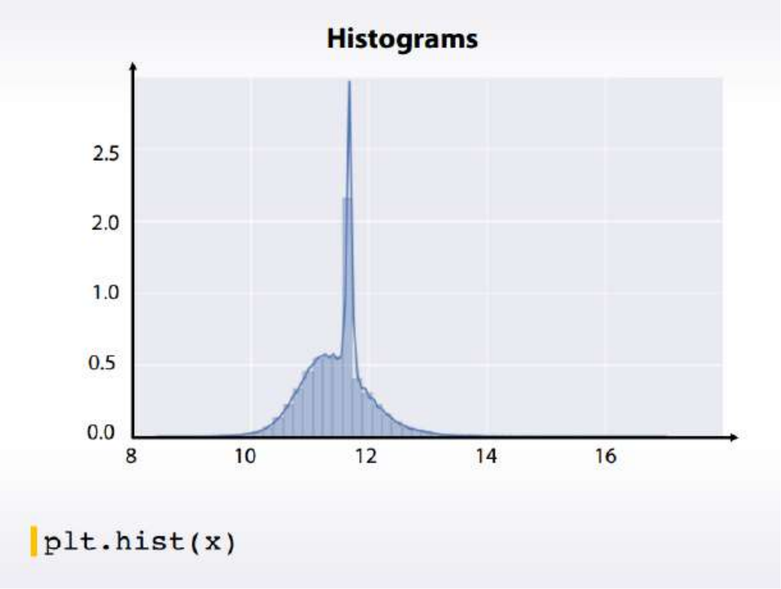
Horizontal lines show repeat value, no vertical lines show shuffled ro



Other tools

```
x.value_counts()
x.isnull()
```

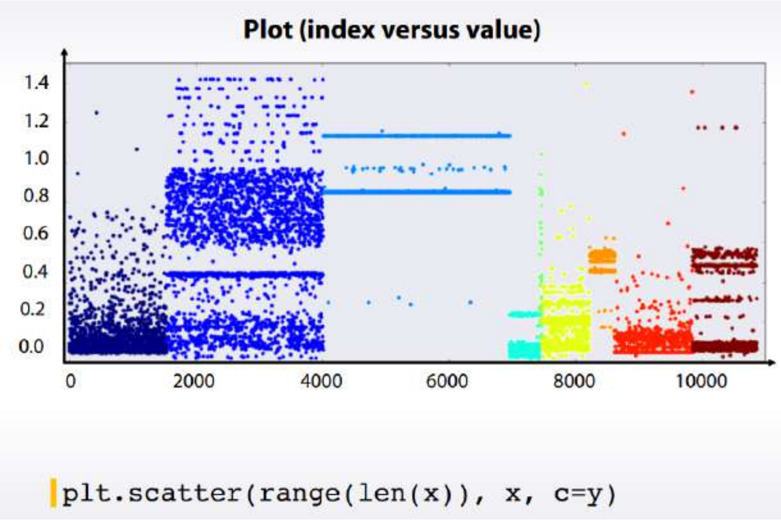
It feels like 0 is there many times, but log leke pata chala asa nahi hai



Histograms

```
plt.hist(x)
```

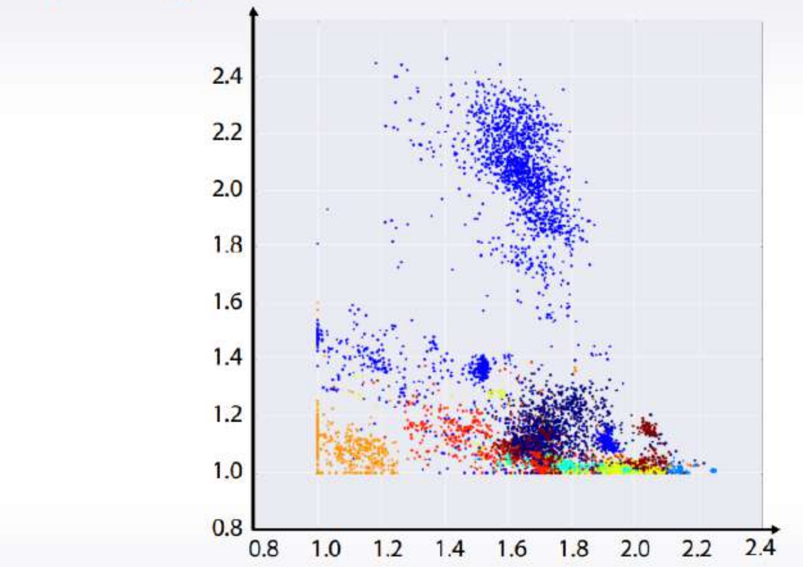
Not shuffled rows



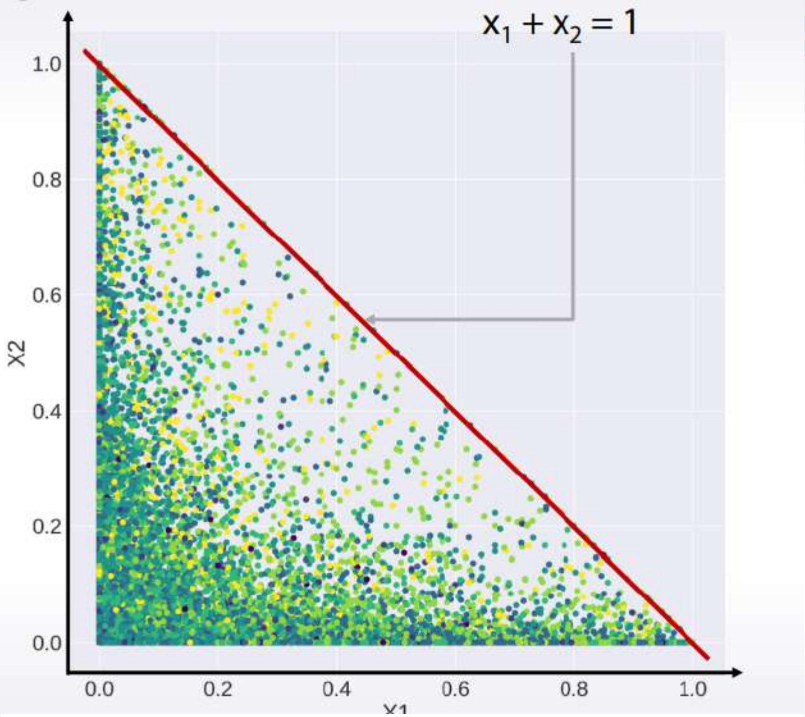
Plot (index versus value)

```
plt.scatter(range(len(x)), x, c=y)
```

Exploring feature relations

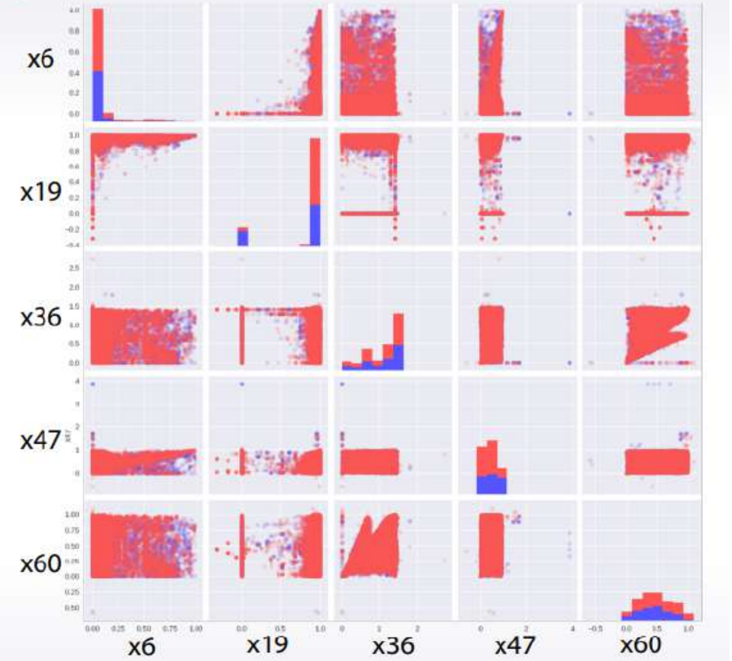


```
plt.scatter(x1, x2)
```

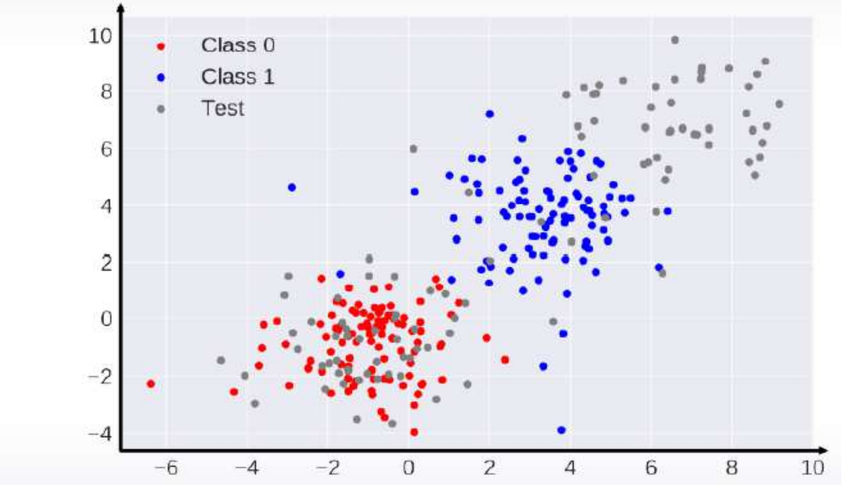


Clearly they follow x2 <- x1. So we should make new feature as difference between x1, x2

Exploring individual features: pairs



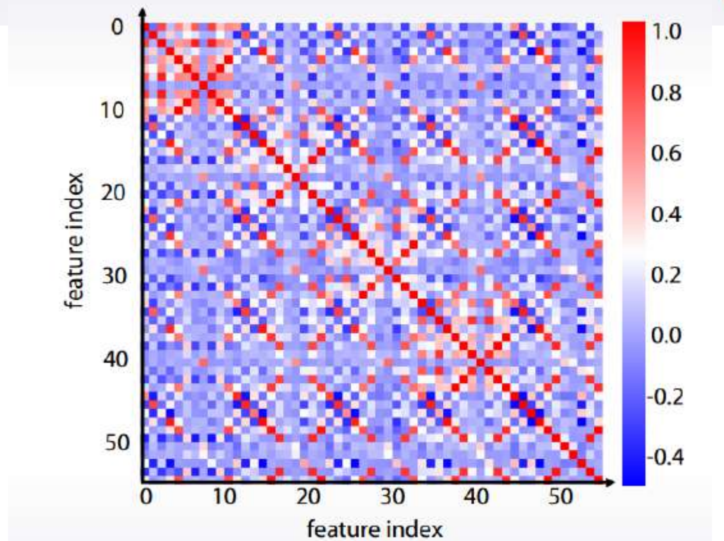
```
pd.scatter_matrix(df)
```



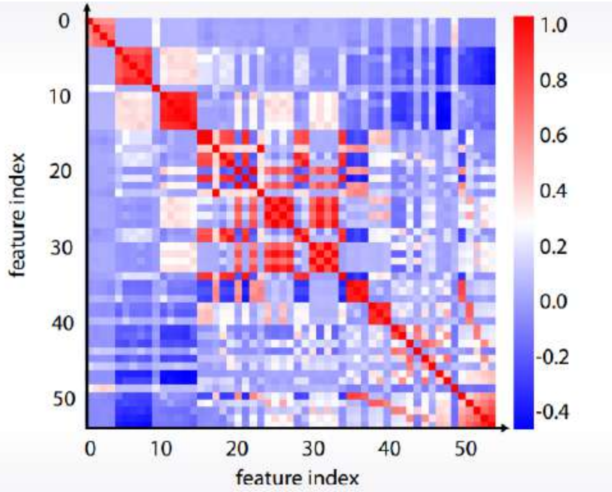
```
plt.scatter(x1, x2)
```

Test in right side is far away from everyone, which is bad
We should check if we have been doing things correctly

Exploring individual features: pairs/groups

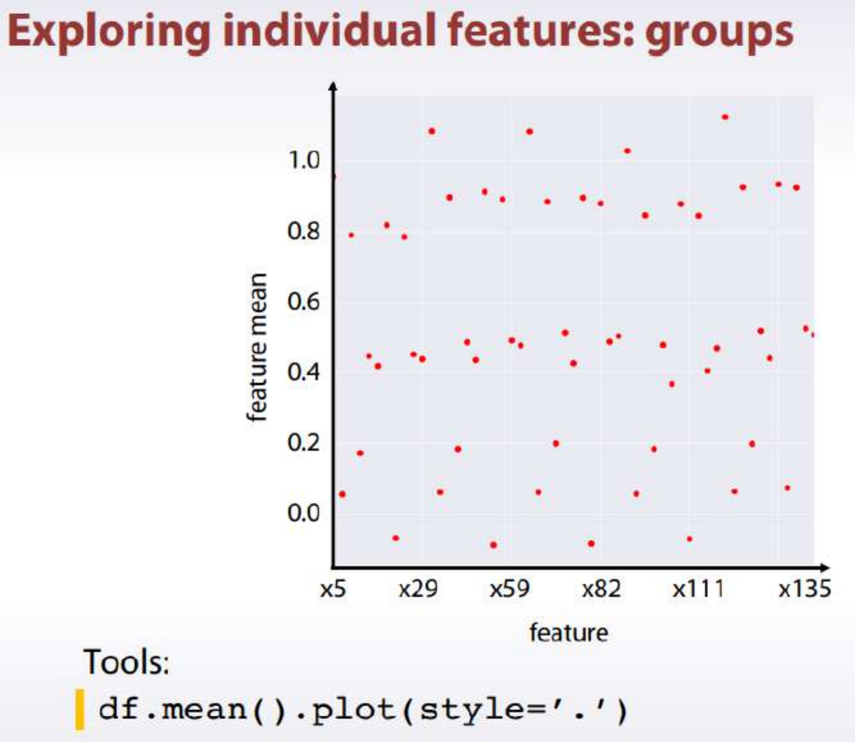


This is same correlation matrix just after running Kmeans so we have feature groups



Tools:

```
df.corr(), plt.matshow( … )
```

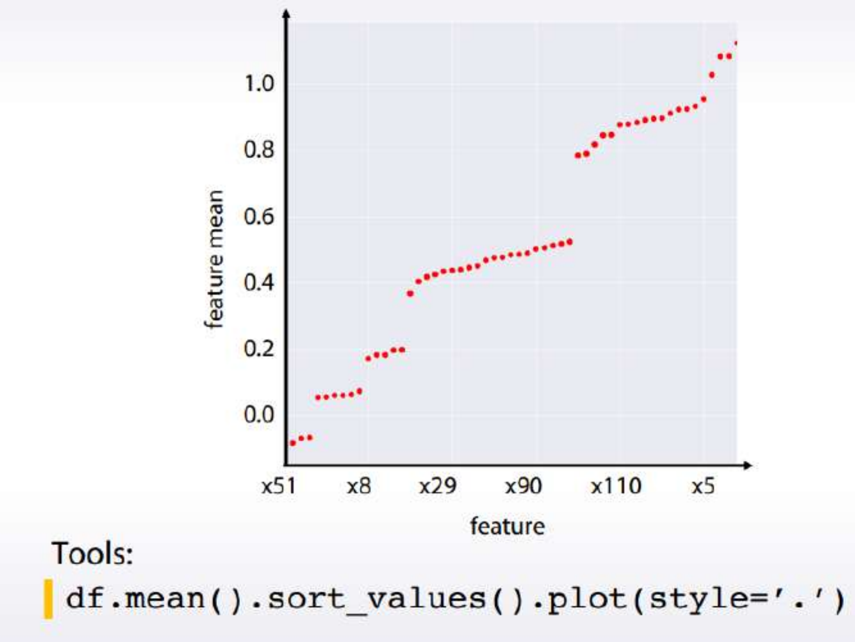


Exploring individual features: groups

Tools:

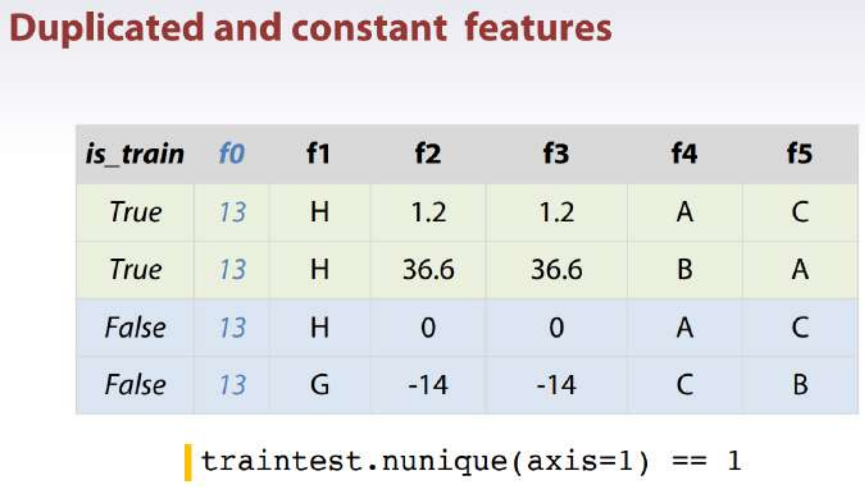
```
df.mean().plot(style='.')
```

Just sort karne se nice relations mil gaye



Tools:

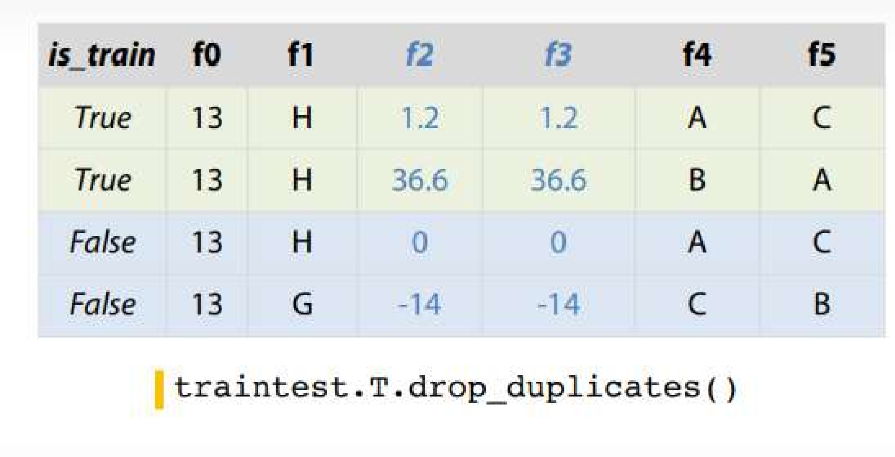
```
df.mean().sort_values().plot(style='.')
```



Duplicated and constant features

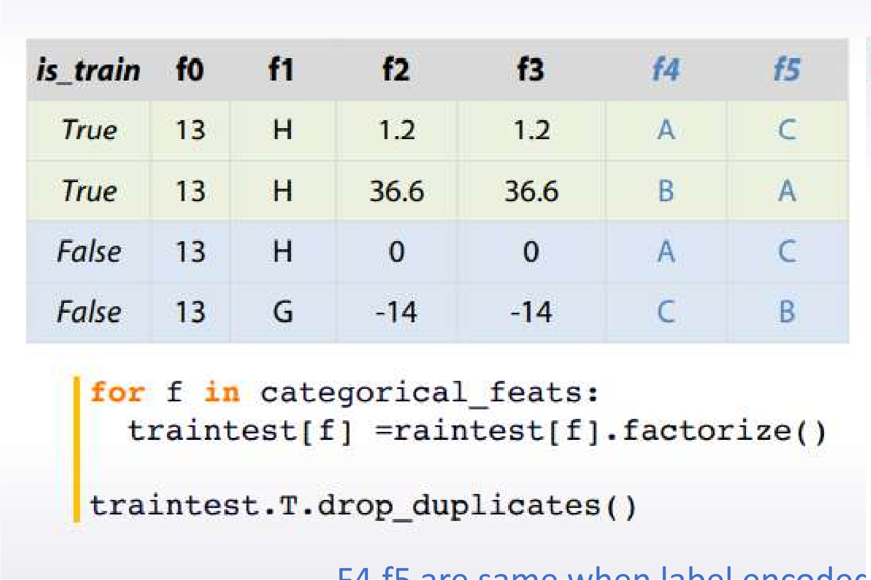
is_train	f0	f1	f2	f3	f4	f5
True	13	H	1.2	1.2	A	C
True	13	H	36.6	36.6	B	A
False	13	H	0	0	A	C
False	13	G	-14	-14	C	B

```
traintest.nunique(axis=1) == 1
```



is_train	f0	f1	f2	f3	f4	f5
True	13	H	1.2	1.2	A	C
True	13	H	36.6	36.6	B	A
False	13	H	0	0	A	C
False	13	G	-14	-14	C	B

```
traintest.T.drop_duplicates()
```

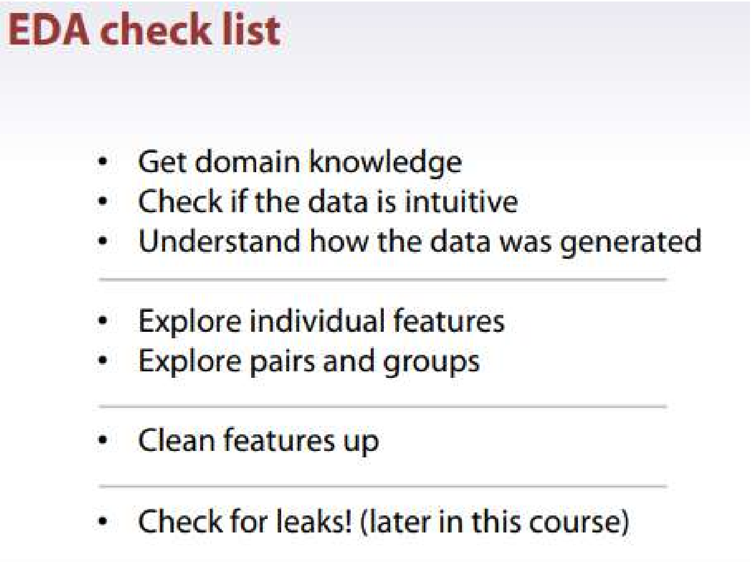


is_train	f0	f1	f2	f3	f4	f5
True	13	H	1.2	1.2	A	C
True	13	H	36.6	36.6	B	A
False	13	H	0	0	A	C
False	13	G	-14	-14	C	B

```
for f in categorical_feats:
  traintest[f] =raintest[f].factorize()

traintest.T.drop_duplicates()
```

F4,f5 are same when label encodec



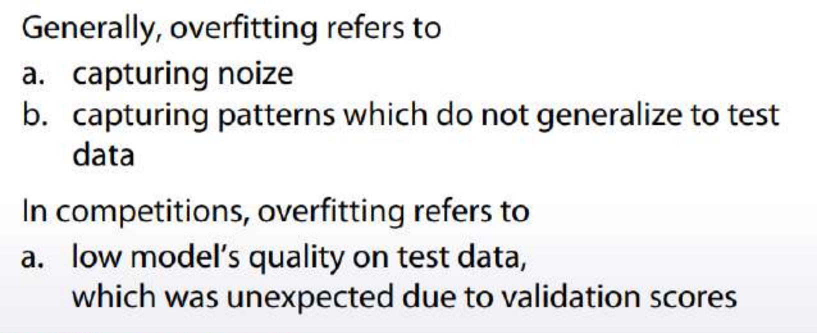
EDA check list

- Get domain knowledge
- Check if the data is intuitive
- Understand how the data was generated

- Explore individual features
- Explore pairs and groups

- Clean features up

- Check for leaks! (later in this course)

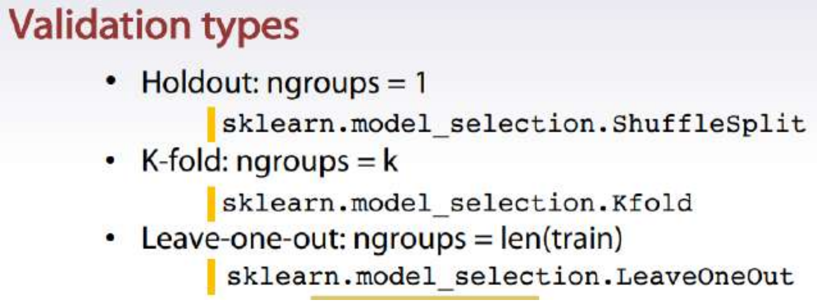


Generally, overfitting refers to

a. capturing noize
b. capturing patterns which do not generalize to test data

In competitions, overfitting refers to

a. low model's quality on test data, which was unexpected due to validation scores

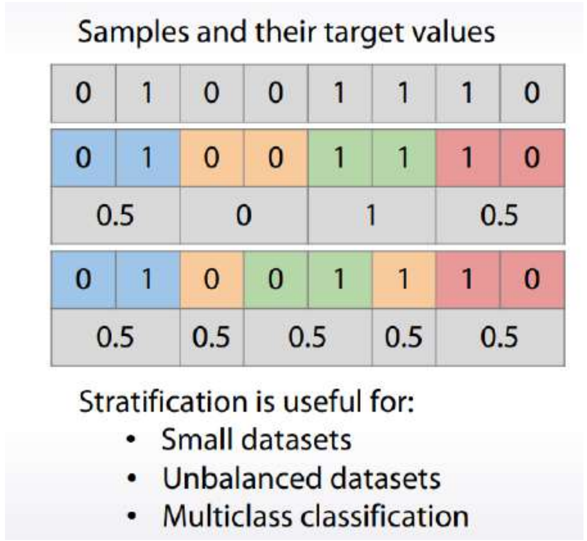


Validation types

- Holdout: ngroups = 1
  `sklearn.model_selection.ShuffleSplit`
- K-fold: ngroups = k
  `sklearn.model_selection.Kfold`
- Leave-one-out: ngroups = len(train)
  `sklearn.model_selection.LeaveOneOut`

Holdout when lot of data, others when less data.
LOO when minimal data

Stratification preserve the same target distribution over different folds



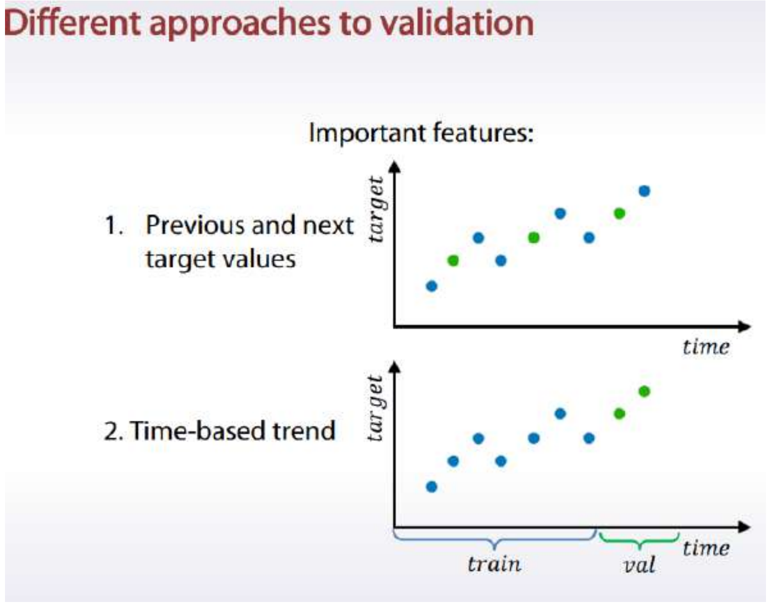
Samples and their target values

0	1	0	0	1	1	1	0
0	1	0	0	1	1	1	0
0.5		0		1		0.5	
0	1	0	0	1	1	1	0
0.5		0.5	0.5		0.5	0.5	

Stratification is useful for:

- Small datasets
- Unbalanced datasets
- Multiclass classification

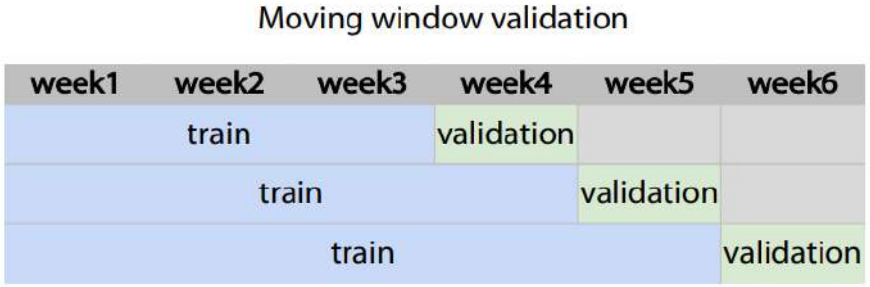
Validation schemes are supposed to be used to estimate quality of the model. When you found the right hyper-parameters and want to get test predictions **don't forget to retrain your model using all training data.**



Different approaches to validation

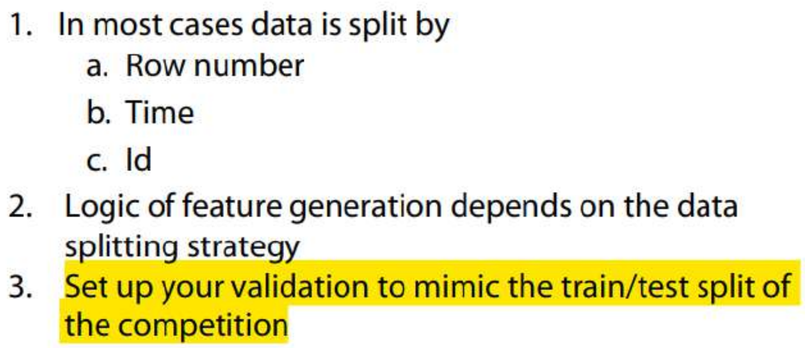
Important features:

1. Previous and next target values
2. Time-based trend



Moving window validation

week1	week2	week3	week4	week5	week6
train			validation		
train				validation	
train					validation



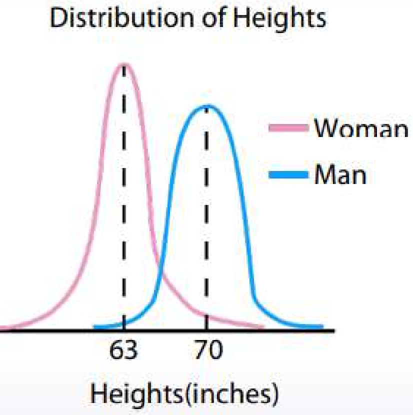
1. In most cases data is split by
   a. Row number
   b. Time
   c. Id
2. Logic of feature generation depends on the data splitting strategy
3. Set up your validation to mimic the train/test split of the competition

Causes of different scores and optimal parameters

1. Too little data
2. Too diverse and inconsistent data

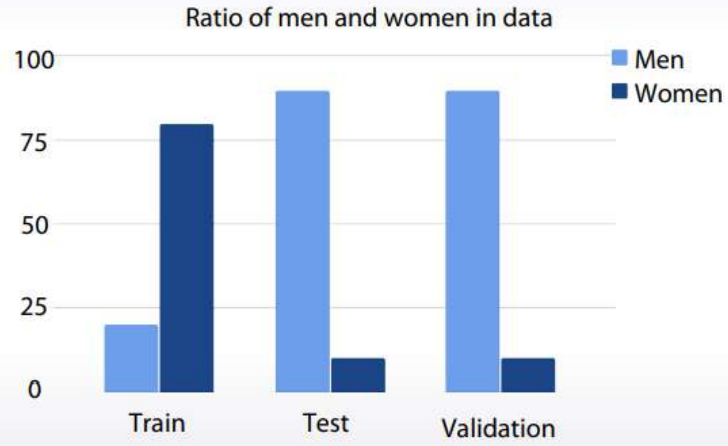
We should do extensive validation

1. Average scores from different KFold splits
2. Tune model on one split, evaluate score on the other

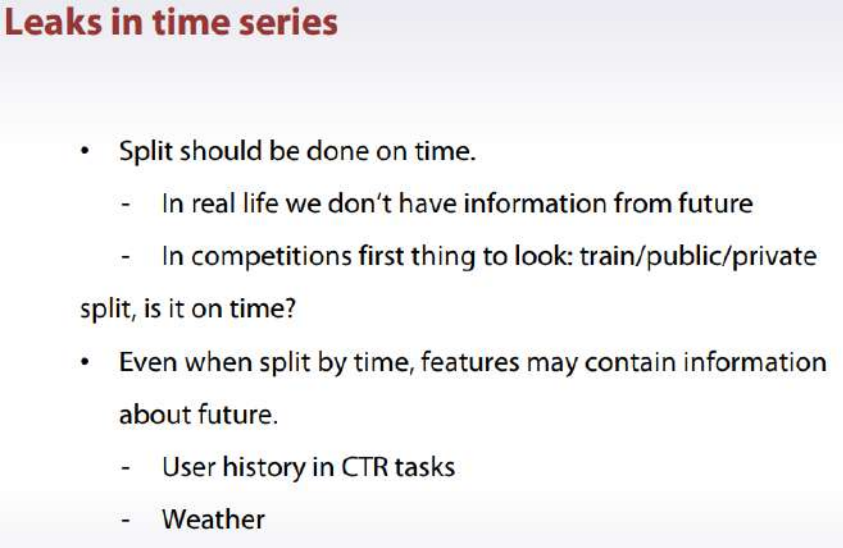


- Mean for train: Calculate from the train data
- Mean for test: Leaderboard probing

Submission stage: different distributions



- If submission's score do not match local validation score, we should
  - Check if we have too little data in public LB
  - Check if we overfitted
  - Check if we chose correct splitting strategy
  - Check if train/test have different distibutions
- Expect LB shuffle because of
  - Randomness
  - Little amount of data
  - Different public/private distributions

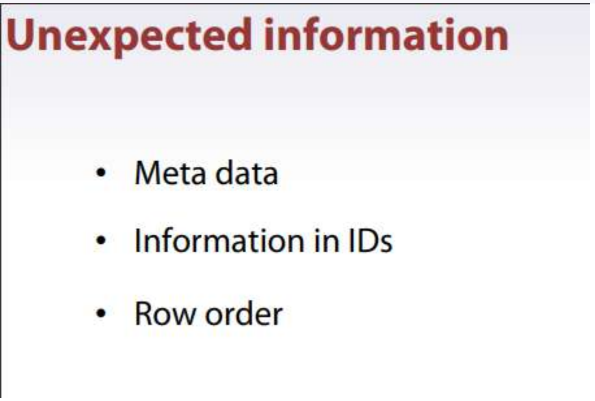


Leaks in time series

- Split should be done on time.
  - In real life we don't have information from future
  - In competitions first thing to look: train/public/private

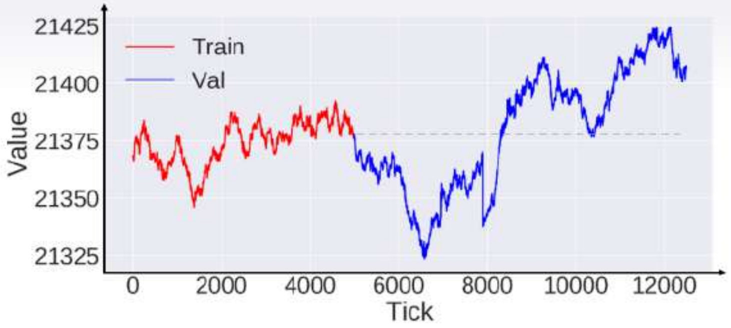
split, is it on time?

- Even when split by time, features may contain information about future.
  - User history in CTR tasks
  - Weather



Unexpected information

- Meta data
- Information in IDs
- Row order



$$Loss(\hat{y}_i; y_i) = \begin{cases} |y_i - \hat{y}_i|, & \text{if trend predicted correctly} \\ (y_i - \hat{y}_i)^2, & \text{if trend predicted incorrectly} \end{cases}$$

Predict *trend* instead of the values:

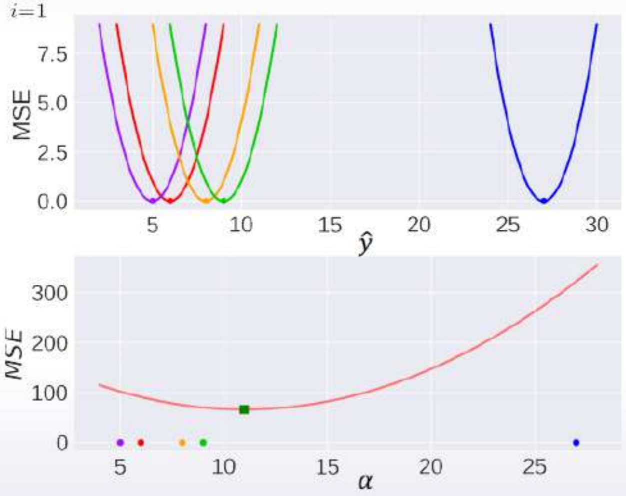
Predict $y_{last} + 10^{-6}$
or $y_{last} - 10^{-6}$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \alpha)^2$$

Best constant: target mean

Data:

X	Y
...	5
...	9
...	8
...	6
...	27

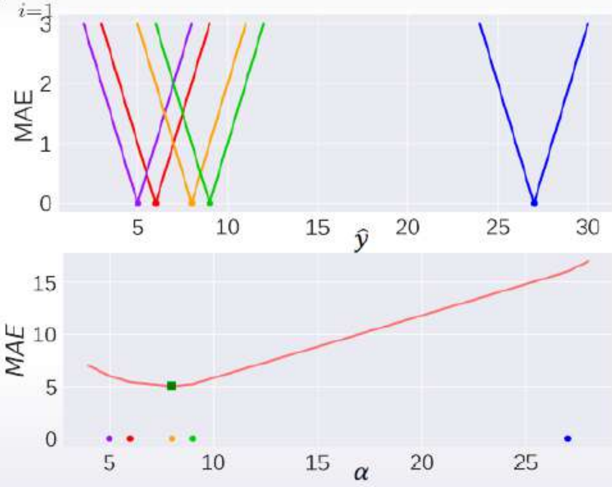


$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \alpha|$$

Best constant: target median

Data:

X	Y
-1	5
1	9
-2	8
3	6
3	27



$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Root mean square error

- $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} = \sqrt{\text{MSE}}$
- $\text{MSE}(a) > \text{MSE}(b) \iff \text{RMSE}(a) > \text{RMSE}(b)$
- $\frac{\partial \text{RMSE}}{\partial \hat{y}_i} = \frac{1}{2\sqrt{MSE}} \frac{\partial \text{MSE}}{\partial \hat{y}_i}$

R-squared:

$$R^2 = 1 - \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2} = 1 - \frac{MSE}{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

- **Do you have outliers in the data?**
  - Use MAE
- **Are you sure they are outliers?**
  - Use MAE
- **Or they are just unexpected values we should still care about?**
  - Use MSE

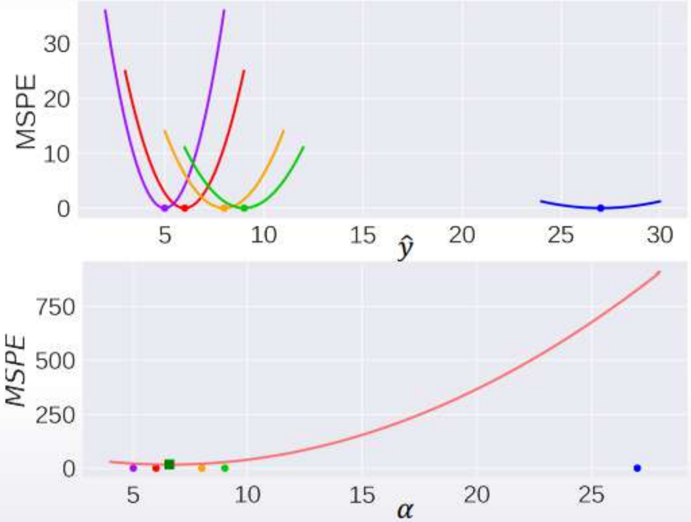
- **MSE, RMSE, R-squared**
  - They are the same from optimization perspective
- **MAE**
  - Robust to outliers

$$\mathrm{MSPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left(\frac{y_i-\alpha}{y_i}\right)^2$$

Best constant:
weighted target mean

Data:

X	Y
-1	4
1	3
-2	6
3	7
3	25



(R)MSLE: Root Mean Square Logarithmic Error

$$\mathrm{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i+1)-\log(\hat{y}_i+1))^2} =$$
$$= RMSE\left(\log(y_i+1), \log(\hat{y}_i+1)\right) =$$
$$= \sqrt{MSE\left(\log(y_i+1), \log(\hat{y}_i+1)\right)}$$

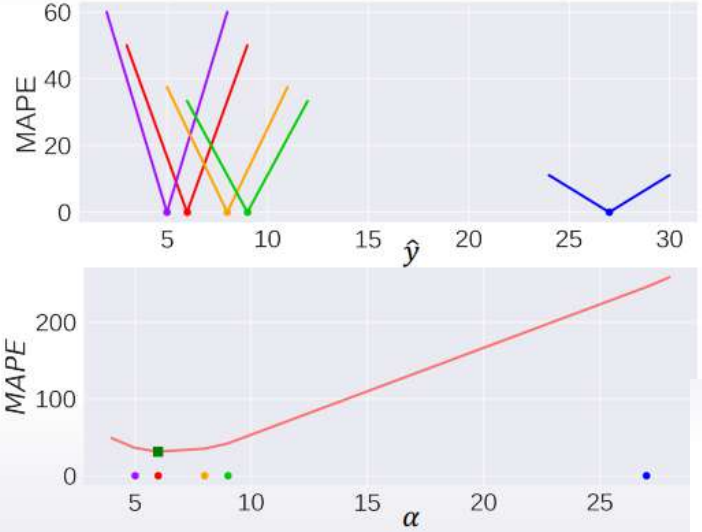
- **(R)MSPE**
  - Weighted version of MSE
- **MAPE**
  - Weighted version of MAE
- **(R)MSLE**
  - MSE in log space

$$\mathrm{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i-\alpha}{y_i}\right|$$

Best constant:
weighted target median

Data:

X	Y
-1	4
1	3
-2	6
3	7
3	25



- Best constant:
  - **set α_i to frequency of i-th class.**

$$\mathrm{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}[\alpha = y_i]$$

- How frequently our class prediction is correct.
- Best constant:
  - **predict the most frequent class.**
- Dataset:
  - 10 cats
  - 90 dogs

Predict always dog:
Accuracy = **0.9**!

- Binary:

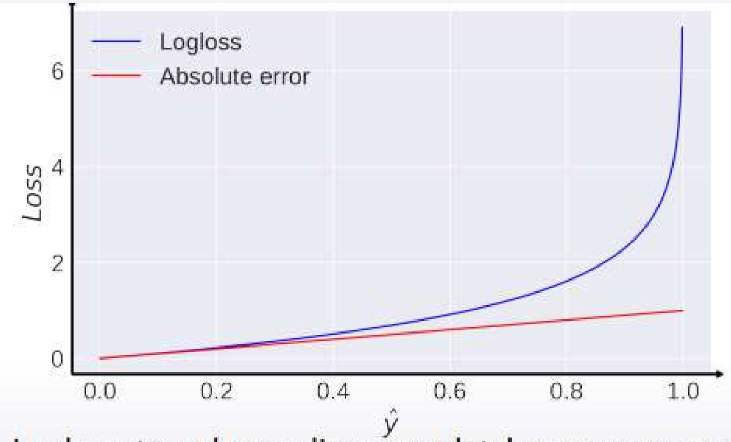
$$\mathrm{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N} y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$
$$y_i \in \mathbb{R}, \quad \hat{y}_i \in \mathbb{R}$$

- Multiclass:

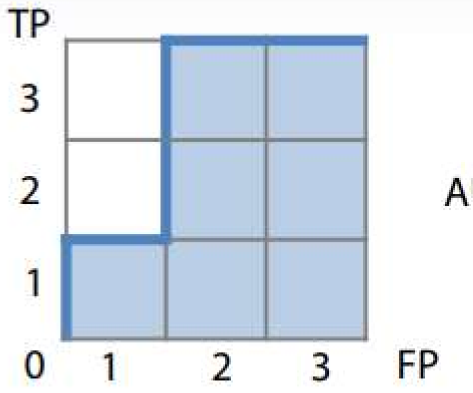
$$\mathrm{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L} y_{il}\log(\hat{y}_{il})$$
$$y_i \in \mathbb{R}^L, \quad \hat{y}_i \in \mathbb{R}^L$$

- In practice:

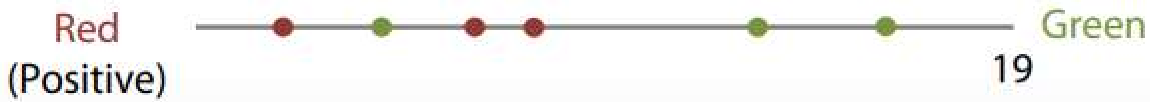
$$\mathrm{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L} y_{il}\log(\min(\max(\hat{y}_{il}, 10^{-15}), 1-10^{-15}))$$



- Logloss strongly penalizes completely wrong answers

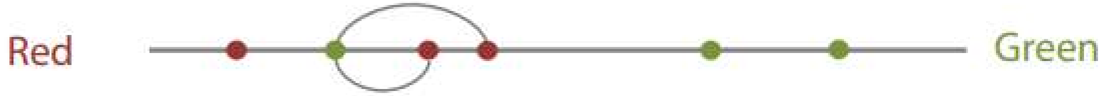


AUC = 7/9



TP – true positives, **FP** – false positives

$$\text{AUC} = \frac{\#\text{ correctly ordered pairs}}{\text{total number of pairs}} = $$
$$= 1 - \frac{\#\text{ incorrectly ordered pairs}}{\text{total number of pairs}}$$



pair = (red object, green object)

Cohen's Kappa motivation

Dataset:

- 10 cats
- 90 dogs

Predict *20 cats* and *80 dogs* at random: *accuracy* ~ 0.74

0.2*0.1 + 0.8*0.9 = 0.74

$$\text{Cohen's Kappa} = 1 - \frac{1 - \text{accuracy}}{1 - p_e}$$

p_e – what accuracy would be on average, if we randomly permute our predictions

$$p_e = \frac{1}{N^2} \sum_k n_{k1} n_{k2}$$

Confusion matrix *C*

pred\ true	cat	dog	tiger
cat	4	2	3
dog	2	88	4
tiger	4	10	12

Weight matrix *W*

pred\ true	cat	dog	tiger
cat	0	1	10
dog	1	0	10
tiger	1	1	0

Linear weights

pred\ true	1	2	3
1	0	1	2
2	1	0	1
3	2	1	0

Quadratic weights

pred\ true	1	2	3
1	0	1	4
2	1	0	1
3	4	1	0

$$\text{weighted error} = \frac{1}{const}\sum_{i,j} C_{ij}W_{ij}$$

$$\boxed{\text{weighted kappa} = 1 - \frac{\text{weighted error}}{\text{weighted baseline error}}}$$

Approaches for target metric optimizatio

- Just run the right model!
  - MSE, Logloss
- Preprocess train and optimize another metri
  - MSPE, MAPE, RMSLE, ...
- Optimize another metric, postprocess predi
  - Accuracy, Kappa
- **Write custom loss function**
  - Any, if you can

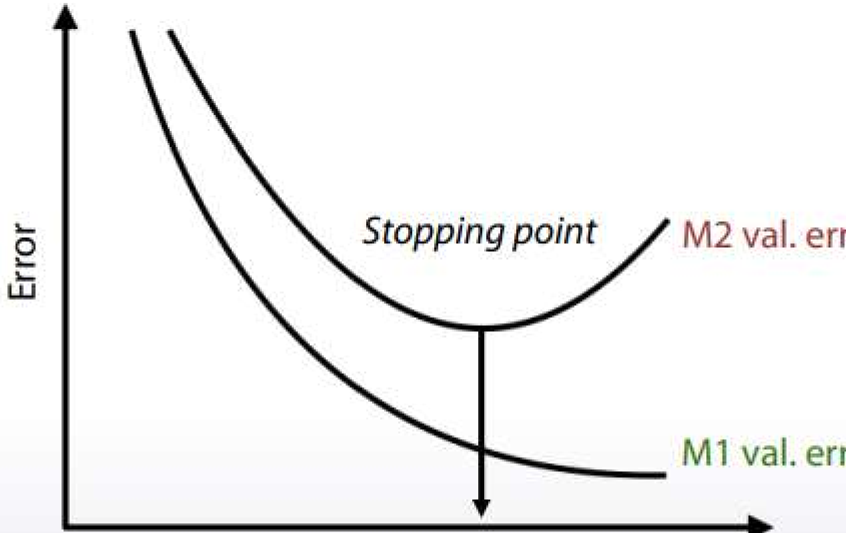
Custom loss for XGBoost

- **Define an 'objective':**
  - function that computes
    derivatives w.r.t. predict

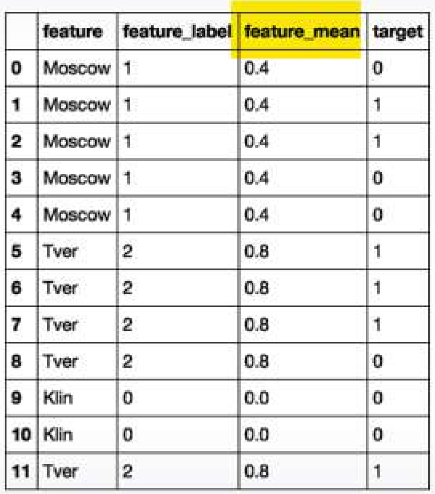
```
def logregobj(pred
    labels = dtrai
    preds = 1.0 /
    grad = preds -
    hess = preds *
    return grad, h
```

Early stopping

- Optimize metric M1, monitor metric M2
  - Stop when M2 score is the best

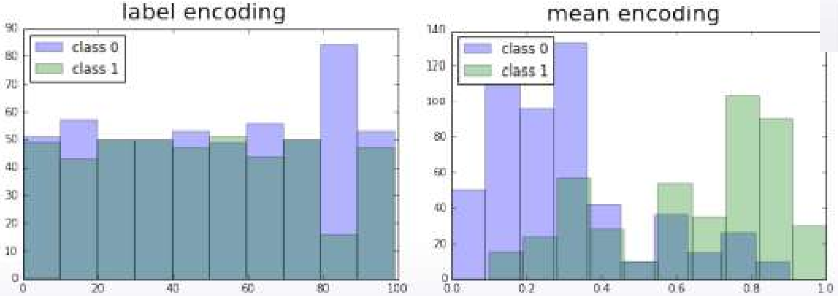


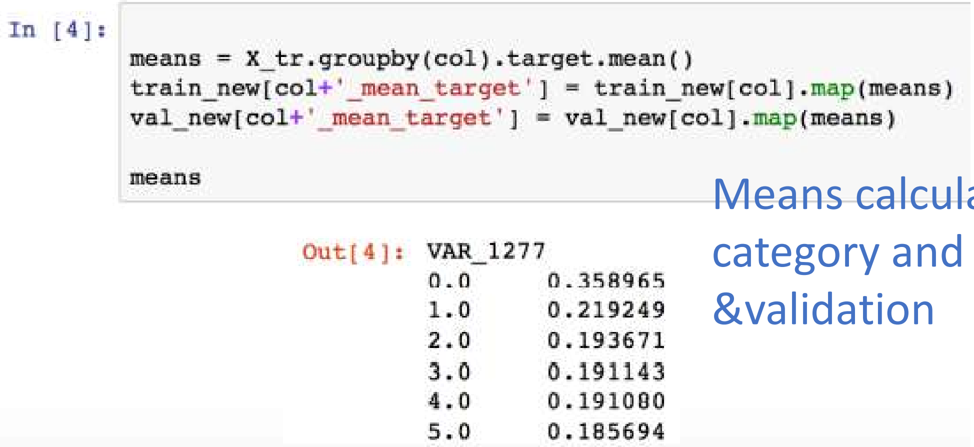
- Categorical feature
  - some city
- Binary classification



	feature	feature_label	feature_mean	target
0	Moscow	1	0.4	0
1	Moscow	1	0.4	1
2	Moscow	1	0.4	1
3	Moscow	1	0.4	0
4	Moscow	1	0.4	0
5	Tver	2	0.8	1
6	Tver	2	0.8	1
7	Tver	2	0.8	1
8	Tver	2	0.8	0
9	Klin	0	0.0	0
10	Klin	0	0.0	0
11	Tver	2	0.8	1

1. Label encoding gives random order. No correlation with target
2. Mean encoding helps to separate zeros from ones





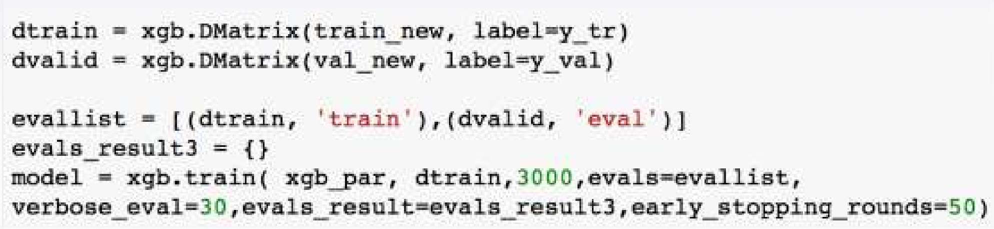
```
In [4]:
means = X_tr.groupby(col).target.mean()
train_new[col+'_mean_target'] = train_new[col].map(means)
val_new[col+'_mean_target'] = val_new[col].map(means)

means
```

```
Out[4]: VAR_1277
        0.0       0.358965
        1.0       0.219249
        2.0       0.193671
        3.0       0.191143
        4.0       0.191080
        5.0       0.185694
```

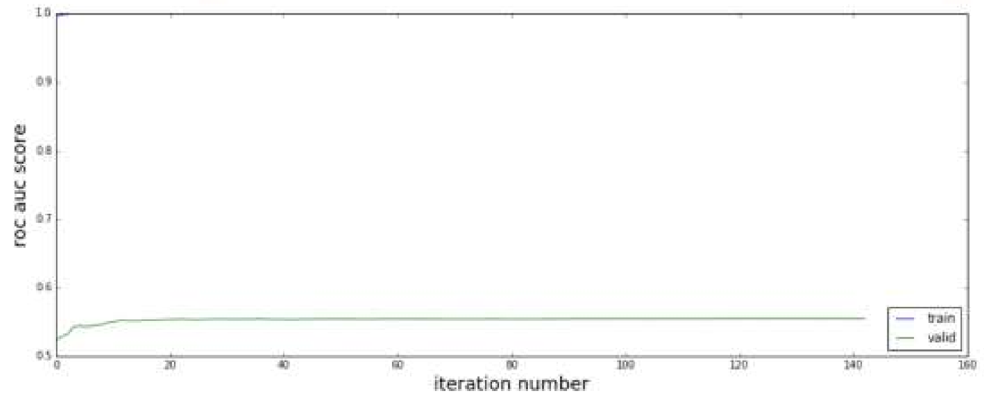
Means calculated for a given category and mapped to training &validation

Springleaf example

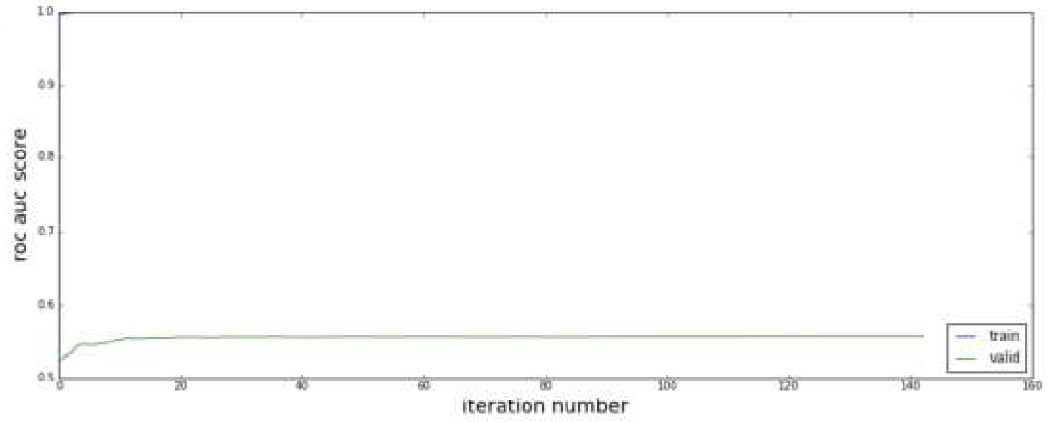


```
dtrain = xgb.DMatrix(train_new, label=y_tr)
dvalid = xgb.DMatrix(val_new, label=y_val)

evallist = [(dtrain, 'train'),(dvalid, 'eval')]
evals_result3 = {}
model = xgb.train( xgb_par, dtrain,3000,evals=evallist,
verbose_eval=30,evals_result=evals_result3,early_stopping_rounds=50)
```



Overfit



Train

	feature	feature_label	feature_mean	target
8	Tver	2	0	0
9	Klin	0	0	0

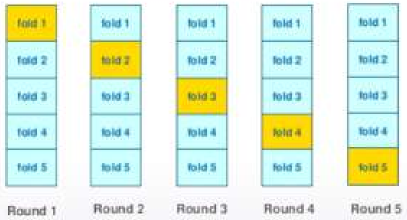
Validation

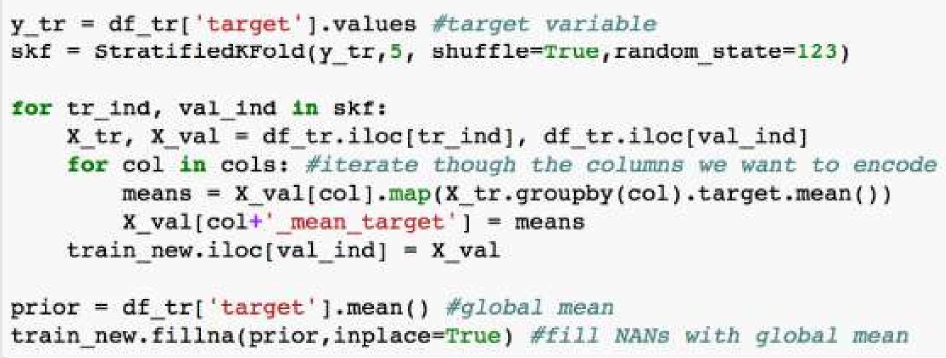
	feature	feature_label	feature_mean	target
10	Klin	0	1	1
11	Tver	2	1	1

Regularization. CV loop

- Robust and intuitive
- Usually decent results with 4-5 folds across different datasets
- Need to be careful with extreme situations like LOO

KFold scheme





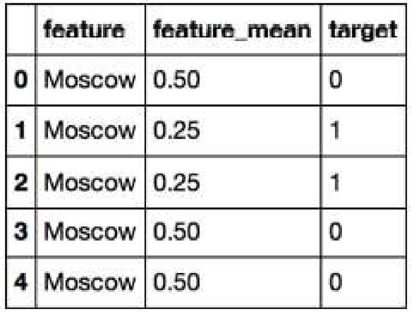
```
y_tr = df_tr['target'].values #target variable
skf = StratifiedKFold(y_tr,5, shuffle=True,random_state=123)

for tr_ind, val_ind in skf:
    X_tr, X_val = df_tr.iloc[tr_ind], df_tr.iloc[val_ind]
    for col in cols: #iterate though the columns we want to encode
        means = X_val[col].map(X_tr.groupby(col).target.mean())
        X_val[col+'_mean_target'] = means
    train_new.iloc[val_ind] = X_val

prior = df_tr['target'].mean() #global mean
train_new.fillna(prior,inplace=True) #fill NANs with global mean
```

- Perfect feature for LOO scheme
- Target variable leakage is still present even for KFold scheme

Leave-one-out



	feature	feature_mean	target
0	Moscow	0.50	0
1	Moscow	0.25	1
2	Moscow	0.25	1
3	Moscow	0.50	0
4	Moscow	0.50	0

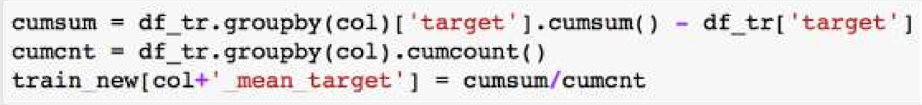
Regularization.Smoothing

- Alpha controls the amount of regularization
- Only works together with some other regularization method

$$\frac{mean(target)*nrows+globalmean*alpha}{nrows+alpha}$$

Regularization. Expanding mean

- Least amount of leakage
- No hyper parameters
- Irregular encoding quality
- Built - in in CatBoost



```
cumsum = df_tr.groupby(col)['target'].cumsum() - df_tr['target']
cumcnt = df_tr.groupby(col).cumcount()
train_new[col+'_mean_target'] = cumsum/cumcnt
```

- There are a lot ways to regularize mean encodings
- Unending battle with target variable leakage
- CV loop or Expanding mean for practical tasks