

AI Demo Class Feature Engineering

Dr. Tran Anh Tuan
VTCA Head of AI

CONTENTS

- Feature Engineering
 - Infuse Domain Knowledge
 - Feature Encoding
 - Interaction Features
 - Feature Representation
 - Binning
 - Practice : <https://www.datacamp.com/community/tutorials/feature-engineering-kaggle>



“Coming up with features is difficult, timeconsuming, requires expert knowledge. “Applied machine learning” is basically feature engineering.”

Andrew Ng



CONTENTS

“At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.”

— Prof. Pedro Domingos



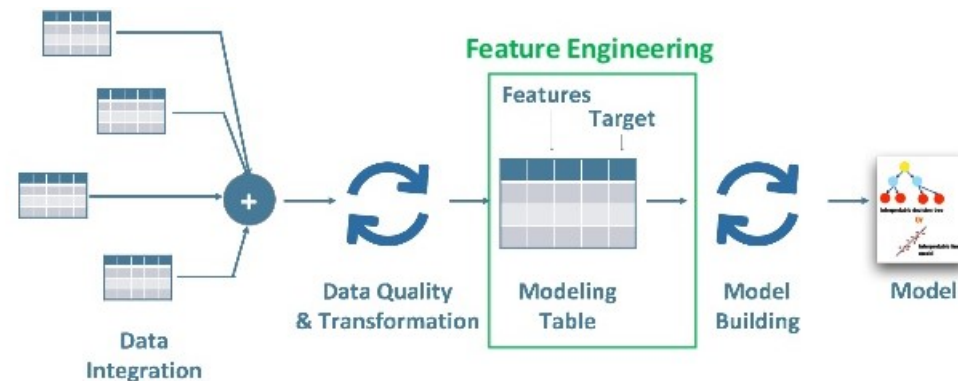
“The algorithms we used are very standard for Kagglers. ... We spent most of our efforts in feature engineering. ... We were also very careful to discard features likely to expose us to the risk of over-fitting our model.”

— Xavier Conort

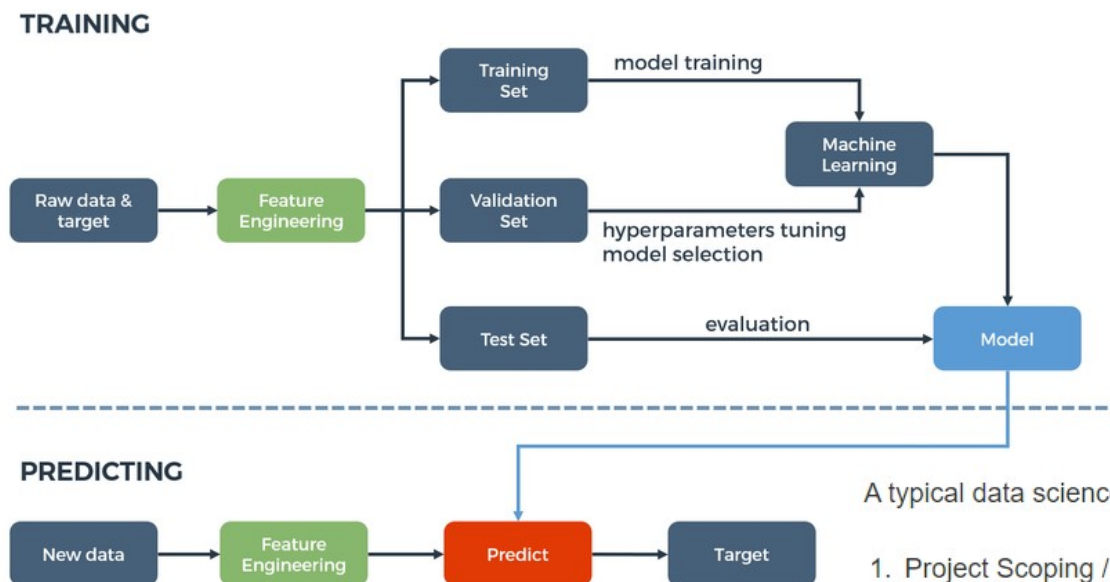


What is Feature Engineering?

- Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in an improved model accuracy on unseen data.
- Features can be engineered by decomposing or splitting features, from external data sources, or aggregating or combining features to create new features.



What is Feature Engineering?

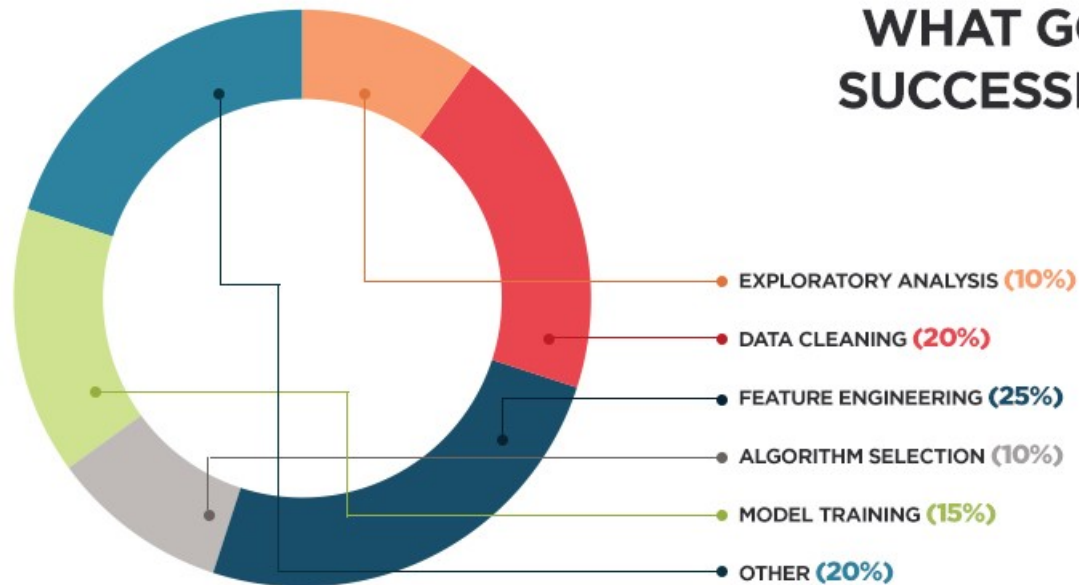


A typical data science process might look like this:

1. Project Scoping / Data Collection
2. Exploratory Analysis
3. Data Cleaning
4. **Feature Engineering**
5. Model Training (including cross-validation to tune hyper-parameters)
6. Project Delivery / Insights

What is Feature Engineering?

WHAT GOES INTO A SUCCESSFUL MODEL

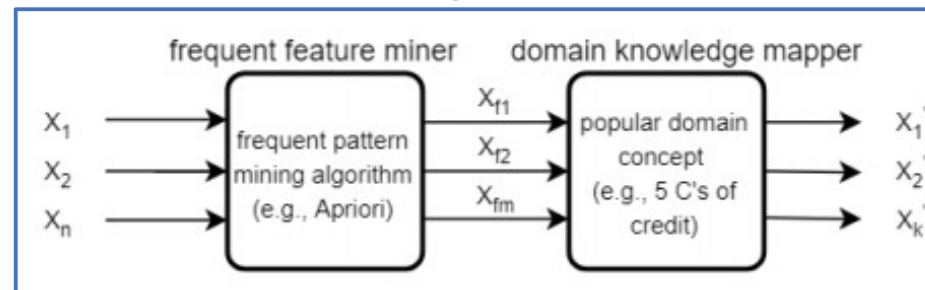


Infuse Domain Knowledge

- You can often engineer informative features by tapping into your (or others') expertise about the domain.
- Try to think of specific information you might want to isolate. Here, you have a lot of "creative freedom."

Feature generalizer

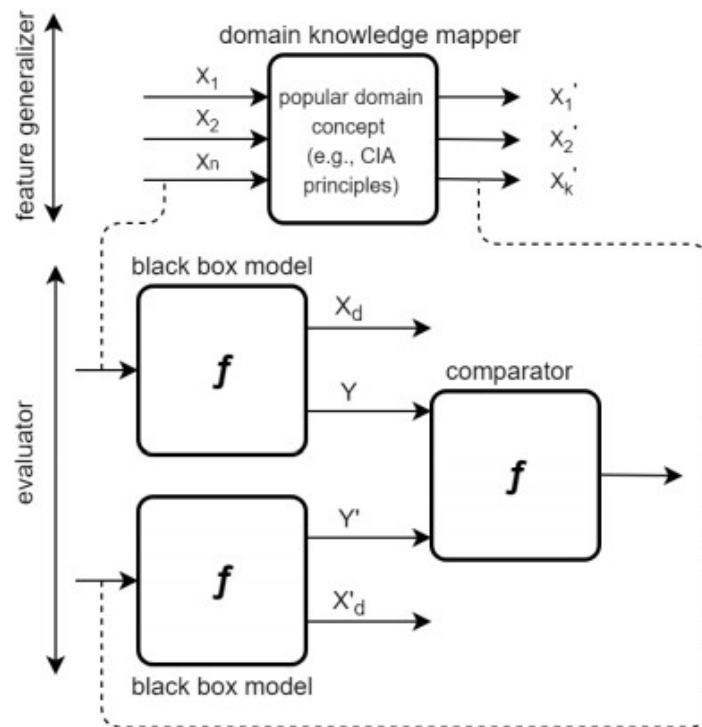
$X_1, X_2, \dots, X_n \subseteq X$ where X is the universal set of features used in mortgage bankruptcy prediction literature



5C's of credit
which refers to
capital, character,
cash flow, conditions,
and collateral.

$X_{f1}, X_{f2}, \dots, X_{fm} \subseteq X$
where X is the
universal set
of features

Infuse Domain Knowledge



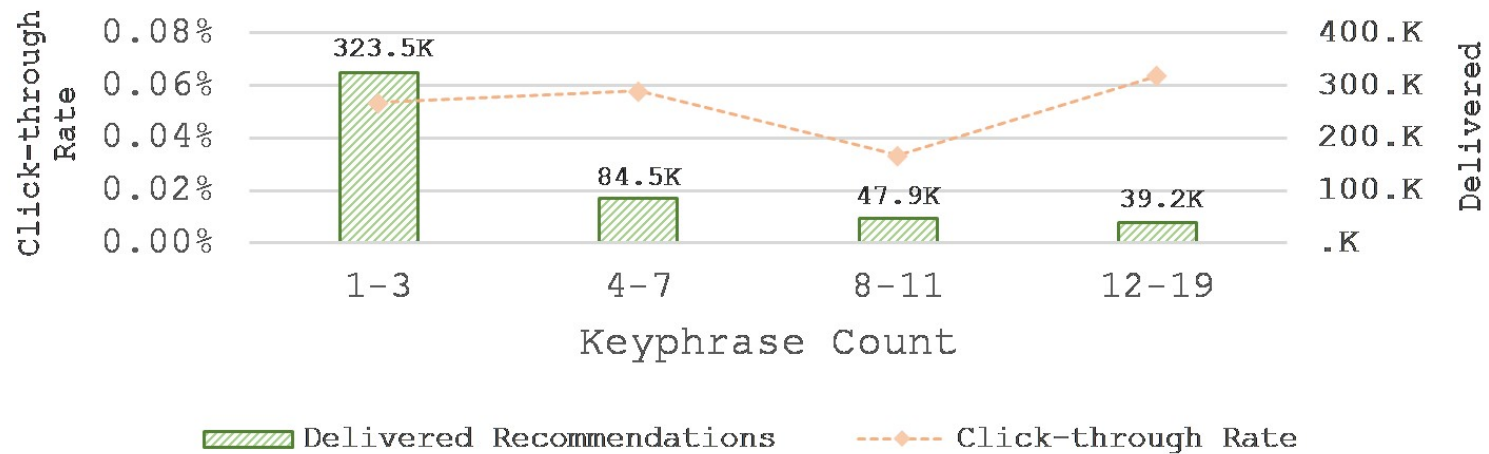
The proposed approach consists of two components: a feature generalizer, which gives a generalized feature set with the help of domain knowledge in two different ways; and an evaluator that produces and compares the results from the "black box" model for multiple configurations of features: domain knowledge infused features, newly constructed features from domain knowledge infused features, selected features, and all features.

CIA principles as domain knowledge, which stands for confidentiality, integrity, and availability.

Infuse Domain Knowledge

- Precise and accurate problem definition is critical for the overall success of a data analysis project. Domain knowledge can often help us reach this precision and accuracy.
- For example, if we want to build a recommender system for an e-commerce platform, we need to understand how users browse online-stores. Without domain knowledge, we might simply define our objective as “building a good recommender system that increases net revenue” which lacks precision.
- However, a domain expert might articulate that when evaluating our recommendation systems, we need to correctly identify the increased user interest generated by recommendations. Thus it may be better to focus on the website CTR (click-through rate), because aside from the recommendations, there can be other reasons behind the revenue lift, such as recent Holiday Sales events

Infuse Domain Knowledge



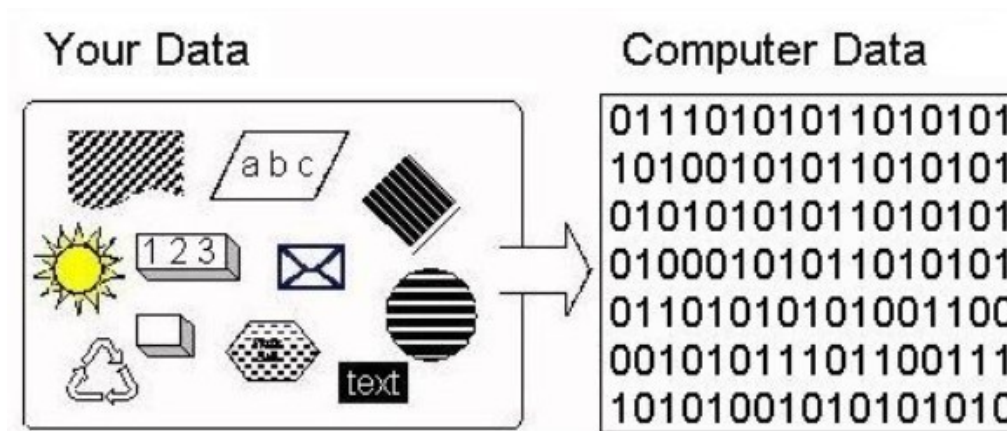
$$\text{CTR} = \frac{\text{Clicks}}{\text{Impressions}} * 100$$

Clicks
number of people
who clicked the ad

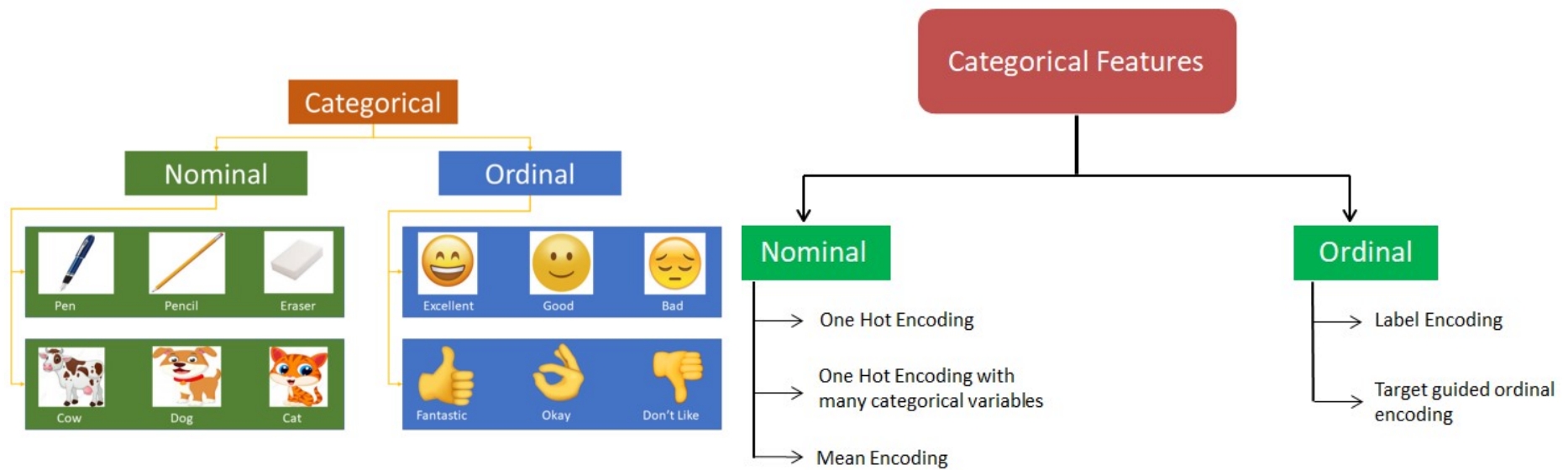
Impressions
number of people
who saw the ad

Feature Encoding

- A big part of the preprocessing is something encoding. This means representing each piece of data in a way that the computer can understand, hence the name encode, which literally means “convert to [computer] code”
- That’s primarily the reason we need to convert categorical columns to numerical columns so that a machine learning algorithm understands it. This process is called categorical encoding.



Feature Encoding



Feature Encoding

- 1) One Hot Encoding
- 2) Label Encoding
- 3) Ordinal Encoding
- 4) Helmert Encoding
- 5) Binary Encoding
- 6) Frequency Encoding
- 7) Mean Encoding

Feature Encoding

- 8) Weight of Evidence Encoding
- 9) Probability Ratio Encoding
- 10) Hashing Encoding
- 11) Backward Difference Encoding
- 12) Leave One Out Encoding
- 13) James-Stein Encoding
- 14) M-estimator Encoding
- 15) Thermometer Encoder

Feature Encoding

- Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.
- One-Hot Encoding is another popular technique for treating categorical variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50




One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

- The problem is that with label encoding, the categories now have natural ordered relationships. The computer does this because it's programmed to treat higher numbers as higher numbers; it will naturally give the higher numbers higher weights. We can see the problem with this in an example:
 - Imagine if you had 3 categories of foods: apples, chicken, and broccoli. Using label encoding, you would assign each of these a number to categorize them: apples = 1, chicken = 2, and broccoli = 3. But now, if your model internally needs to calculate the average across categories, it might do $1 + 3 = 4 / 2 = 2$. This means that according to your model, the average of apples and chicken together is broccoli.

Data		Label Encoding		One Hot Encoding			
Brand	price	Brand	price	Brand_Apple	Brand_Nokia	Brand_Samsung	price
Nokia	10K	1	10K	0	1	0	10K
Samsung	25K	2	25K	0	0	1	25K
Apple	5K	3	5K	1	0	0	5K

	Cat	Dog	Zebra
	1	0	0
	0	1	0
	0	0	1

	Temperature	Color	Target
0	Hot	Red	1
1	Cold	Yellow	1
2	Very Hot	Blue	1
3	Warm	Blue	0
4	Hot	Red	1
5	Warm	Yellow	0
6	Warm	Red	1
7	Hot	Yellow	0
8	Hot	Yellow	1
9	Cold	Yellow	1

- When to use a Label Encoding vs. One Hot Encoding
- We apply One-Hot Encoding when:
 1. The categorical feature is not ordinal (like the countries above)
 2. The number of categorical features is less so one-hot encoding can be effectively applied
- We apply Label Encoding when:
 1. The categorical feature is ordinal (like Jr. kg, Sr. kg, Primary school, high school)
 2. The number of categories is quite large as one-hot encoding can lead to high memory consumption

Feature Encoding

- We do **Ordinal encoding** to ensure the encoding of variables retains the ordinal nature of the variable. This is reasonable only for ordinal variables
- This encoding looks almost similar to Label Encoding but slightly different as Label coding would not consider whether variable is ordinal or not and it will assign sequence of integers
- If we consider in the temperature scale as the order, then the ordinal value should from cold to “Very Hot. “ Ordinal encoding will assign values as (Cold(1) < Warm(2) < Hot(3) < “Very Hot(4)). Usually, we Ordinal Encoding is done starting from 1.

	Temperature	Color	Target	Temp_Ordinal
0	Hot	Red	1	3
1	Cold	Yellow	1	1
2	Very Hot	Blue	1	4
3	Warm	Blue	0	2
4	Hot	Red	1	3
5	Warm	Yellow	0	2
6	Warm	Red	1	2
7	Hot	Yellow	0	3
8	Hot	Yellow	1	3
9	Cold	Yellow	1	1

- In **Helmert Encoding**, the mean of the dependent variable for a level is compared to the mean of the dependent variable over all previous levels.
- This type of encoding is useful when the levels of the categorical variable are ordered in a meaningful way. For example, if we had a categorical variable in which work-related stress was coded as low, medium or high, then comparing the means of the previous levels of the variable would make more sense.

If there are L levels then the first comparison is of level vs. $(L - 1)$ other levels. The weights are then $(L - 1)/L$ for the first level and $-1/L$ for each of the other levels. In your case $L = 4$ so the weights are .75 and -.25 (3 times).

The next comparison has only $L - 1$ levels (the first level is no longer part of the comparisons), so now the weights are $(L - 2)/(L - 1)$ for the first level and $-1/(L - 1)$ for the others (in your case, $2/3$ and $-1/3$. And so on.

Feature Encoding

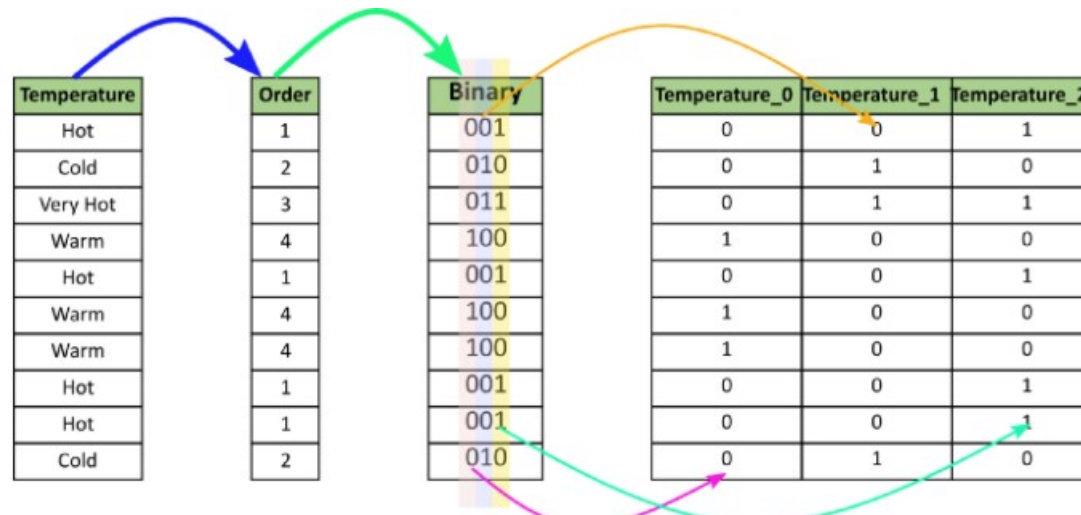
Level of race	New variable 1 (x1)	New variable 2 (x2)	New variable 3 (x3)
	Level 1 v. Later	Level 2 v. Later	Level 3 v. Later
1 (Hispanic)	.75	0	0
2 (Asian)	-.25	.666	0
3 (African American)	-.25	-.333	.5
4 (white)	-.25	-.333	-.5

- **Binary encoding** converts a category into binary digits. Each binary digit creates one feature column. If there are n unique categories, then binary encoding results in the only $\log(\text{base } 2)^n$ features.
- In this example, we have four features; thus, the total number of the binary encoded features will be three features.
- Compared to One Hot Encoding, this will require fewer feature columns (for 100 categories One Hot Encoding will have 100 features while for Binary encoding, we will need just seven features).

Feature Encoding

For Binary encoding, one has to follow the following steps:

- The categories are first converted to numeric order starting from 1 (order is created as categories appear in a dataset and do not mean any ordinal nature)
- Then those integers are converted into binary code, so for example 3 becomes 011, 4 becomes 100
- Then the digits of the binary number form separate columns.



It is a way to utilize the frequency of the categories as labels. In the cases where the frequency is related somewhat with the target variable, it helps the model to understand and assign the weight in direct and inverse proportion, depending on the nature of the data.

Three-step for this :

- Select a categorical variable you would like to transform
- Group by the categorical variable and obtain counts of each category
- Join it back with the training dataset

Feature Encoding

```
fe= df.groupby('Temperature').size()/len(df)
df.loc[:, 'Temp_freq_encode'] = df['Temperature'].map(fe)
df
```

	Temperature	Color	Target	Temp_freq_encode
0	Hot	Red	1	0.4
1	Cold	Yellow	1	0.2
2	Very Hot	Blue	1	0.1
3	Warm	Blue	0	0.3
4	Hot	Red	1	0.4
5	Warm	Yellow	0	0.3
6	Warm	Red	1	0.3
7	Hot	Yellow	0	0.4
8	Hot	Yellow	1	0.4
9	Cold	Yellow	1	0.2

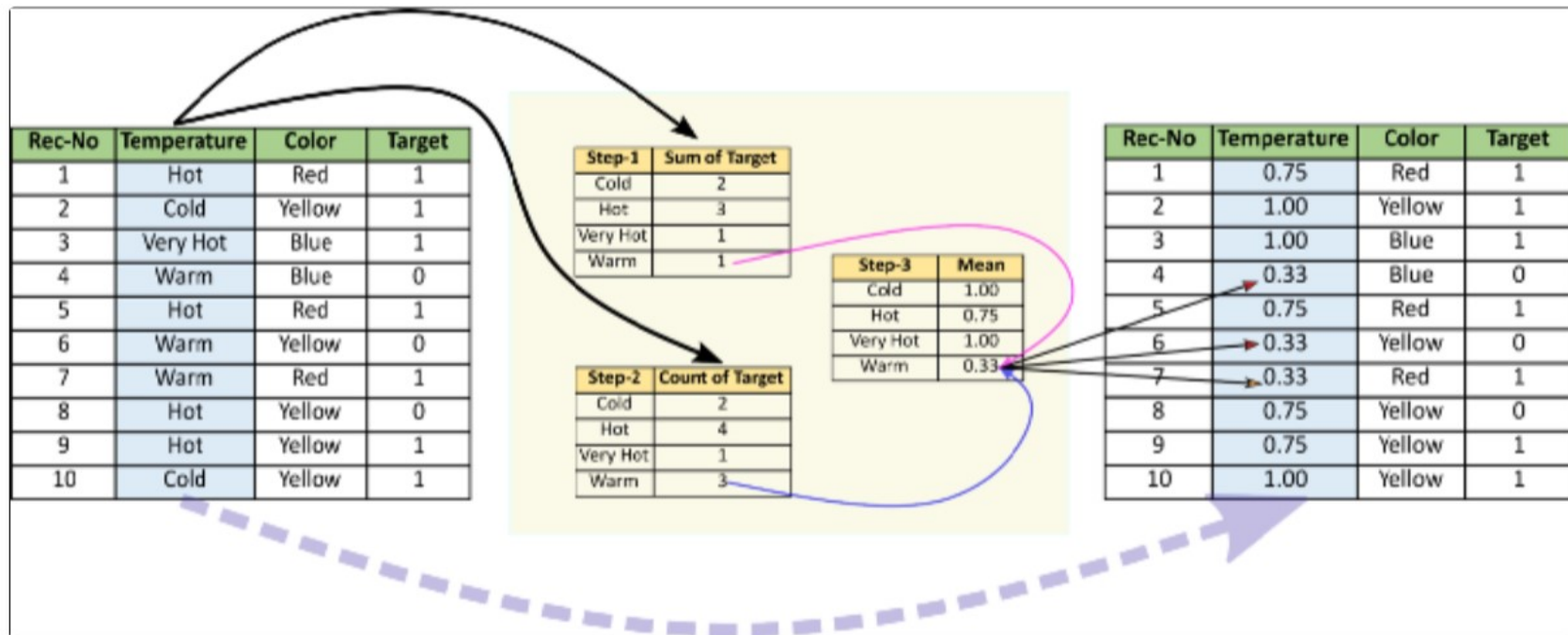
- **Mean encoding** is similar to label encoding, except here labels are correlated directly with the target. For example, in mean target encoding for each category in the feature label is decided with the mean value of the target variable on a training data. This encoding method brings out the relation between similar categories, but the connections are bounded within the categories and target itself.
- The advantages of the mean target encoding are that it does not affect the volume of the data and helps in faster learning.

Feature Encoding

Usually, **Mean encoding** is notorious for over-fitting; thus, a regularization with cross-validation or some other approach is a must on most occasions. Mean encoding approach is as below:

- 1. Select a categorical variable you would like to transform
- 2. Group by the categorical variable and obtain aggregated sum over the “Target” variable. (total number of 1’s for each category in ‘Temperature’)
- 3. Group by the categorical variable and obtain aggregated count over “Target” variable
- 4. Divide the step 2 / step 3 results and join it back with the train.

Feature Encoding



Mean Encoding

Feature Encoding

- **Mean encoding** can embody the target in the label, whereas label encoding does not correlate with the target.
- In the case of a large number of features, mean encoding could prove to be a much simpler alternative. Mean encoding tends to group the classes, whereas the grouping is random in case of label encoding.

Feature Encoding

- Weight of Evidence (WoE) is a measure of the “strength” of a grouping technique to separate good and bad. This method was developed primarily to build a predictive model to evaluate the risk of loan default in the credit and financial industry.
- Weight of evidence (WOE) is a measure of how much the evidence supports or undermines a hypothesis.

It is computed as below:

$$WoE = \left[\ln \left(\frac{Distr\ Goods}{Distr\ Bads} \right) \right] * 100$$

WoE will be 0 if the $P(Goods) / P(Bads) = 1$. That is if the outcome is random for that

group. If $P(Bads) > P(Goods)$ the odds ratio will be < 1 and the WoE will be < 0 ; if, on the other hand, $P(Goods) > P(Bads)$ in a group, then WoE > 0 .

- WoE is well suited for Logistic Regression because the Logit transformation is simply the log of the odds, i.e., $\ln(P(\text{Goods})/P(\text{Bads}))$.
- Therefore, by using WoE-coded predictors in Logistic Regression, the predictors are all prepared and coded to the same scale.
- The parameters in the linear logistic regression equation can be directly compared.

- The WoE transformation has (at least) three advantage:
- 1) It can transform an independent variable so that it establishes a monotonic relationship to the dependent variable. It does more than this — to secure monotonic relationship it would be enough to “recode” it to any ordered measure (for example 1,2,3,4...), but the WoE transformation orders the categories on a “logistic” scale which is natural for Logistic Regression
- 2) For variables with too many (sparsely populated) discrete values, these can be grouped into categories (densely populated), and the WoE can be used to express information for the whole category
- 3) The (univariate) effect of each category on the dependent variable can be compared across categories and variables because WoE is a standardized value (for example you can compare WoE of married people to WoE of manual workers)

Feature Encoding

- It also has (at least) three drawbacks:
- 1) Loss of information (variation) due to binning to a few categories
- 2) It is a “univariate” measure, so it does not take into account the correlation between independent variables
- 3) It is easy to manipulate (over-fit) the effect of variables according to how categories are created

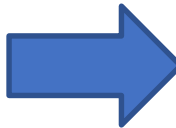
	Good	Bad	WoE
Temperature			
Cold	1.000000	0.000001	13.815511
Hot	0.750000	0.250000	1.098612
Very Hot	1.000000	0.000001	13.815511
Warm	0.333333	0.666667	-0.693147

	Temperature	Color	Target	WoE_Encode
0	Hot	Red	1	1.098612
1	Cold	Yellow	1	13.815511
2	Very Hot	Blue	1	13.815511
3	Warm	Blue	0	-0.693147
4	Hot	Red	1	1.098612
5	Warm	Yellow	0	-0.693147
6	Warm	Red	1	-0.693147
7	Hot	Yellow	0	1.098612
8	Hot	Yellow	1	1.098612
9	Cold	Yellow	1	13.815511

- **Probability Ratio Encoding** is similar to Weight Of Evidence(WoE), with the only difference is the only ratio of good and bad probability is used. For each label, we calculate the mean of target=1, that is the probability of being 1 ($P(1)$), and also the probability of the target=0 ($P(0)$).
- And then, we calculate the ratio $P(1)/P(0)$ and replace the labels by that ratio. We need to add a minimal value with $P(0)$ to avoid any divide by zero scenarios where for any particular category, there is no target=0.

Feature Encoding

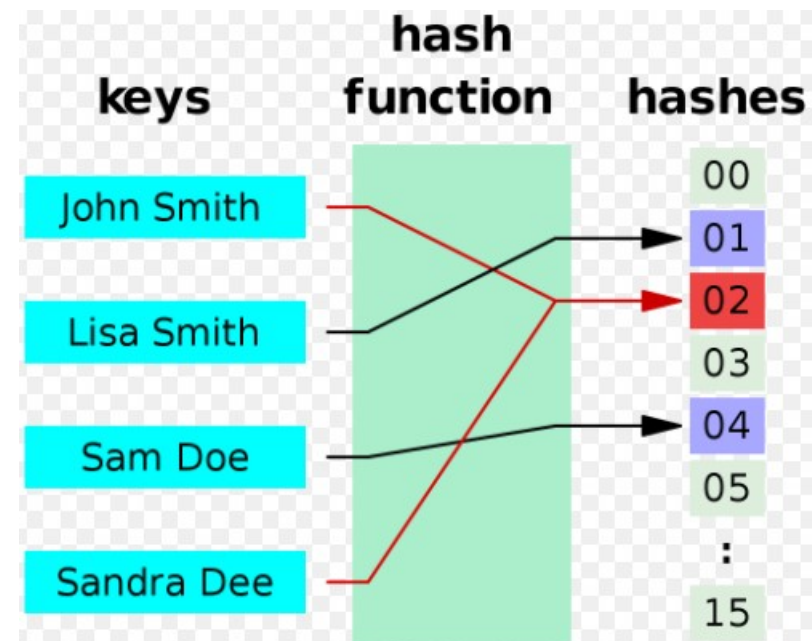
	Good	Bad	PR
Temperature			
Cold	1.000000	0.000001	1000000.0
Hot	0.750000	0.250000	3.0
Very Hot	1.000000	0.000001	1000000.0
Warm	0.333333	0.666667	0.5



	Temperature	Color	Target	PR_Encode
0	Hot	Red	1	3.0
1	Cold	Yellow	1	1000000.0
2	Very Hot	Blue	1	1000000.0
3	Warm	Blue	0	0.5
4	Hot	Red	1	3.0
5	Warm	Yellow	0	0.5
6	Warm	Red	1	0.5
7	Hot	Yellow	0	3.0
8	Hot	Yellow	1	3.0
9	Cold	Yellow	1	1000000.0

Feature Encoding

- Hashing converts categorical variables to a higher dimensional space of integers, where the distance between two vectors of categorical variables is approximately maintained in the transformed numerical dimensional space.
- With Hashing, the number of dimensions will be far less than the number of dimensions with encoding like One Hot Encoding. This method is advantageous when the cardinality of categorical is very high.



Interaction Features

Have you ever heard the phrase, “the sum is greater than the parts?” Well, some features can be combined to provide more information than they would as individuals.

- **Sum of two features:** Let’s say you wish to predict revenue based on preliminary sales data. You have the features *sales_blue_pens* and *sales_black_pens*. You could sum those features if you only care about overall *sales_pens*.
- **Difference between two features:** You have the features *house_built_date* and *house_purchase_date*. You can take their difference to create the feature *house_age_at_purchase*.
- **Product of two features:** You’re running a pricing test, and you have the feature price and an indicator variable *conversion*. You can take their product to create the feature earnings.
- **Quotient of two features:** You have a dataset of marketing campaigns with the features *n_clicks* and *n_impressions*. You can divide clicks by impressions to create *click_through_rate*, allowing you to compare across campaigns of different volume.

Feature Representations

Your data won't always come in the ideal format. You should consider if you'd gain information by representing the same feature in a different way.

- **Date and time features:** Let's say you have the feature *purchase_datetime*. It might be more useful to extract *purchase_day_of_week* and *purchase_hour_of_day*. You can also aggregate observations to create features such as *purchases_over_last_30_days*.
- **Numeric to categorical mappings:** You have the feature *years_in_school*. You might create a new feature grade with classes such as "*Elementary School*", "*Middle School*", and "*High School*".
- **Grouping sparse classes:** You have a feature with many classes that have low sample counts. You can try grouping similar classes and then grouping the remaining ones into a single "*Other*" class.
- **Creating dummy variables:** Depending on your machine learning implementation, you may need to manually transform categorical features into dummy variables. You should always do this *after* grouping sparse classes.

External Data

An underused type of feature engineering is bringing in external data. This can lead to some of the biggest breakthroughs in performance.

- **Time series data:** The nice thing about time series data is that you only need one feature, some form of `date`, to layer in features from another dataset.
- **External API's:** There are plenty of API's that can help you create features. For example, the [Microsoft Computer Vision](#) API can return the number of faces from an image.
- **Geocoding:** Let's say have you `street_address`, `city`, and `state`. Well, you can [geocode](#) them into `latitude` and `longitude`. This will allow you to calculate features such as local demographics (e.g. `median_income_within_2_miles`) with the help of [another dataset](#).
- **Other sources of the same data:** How many ways could you track a Facebook ad campaign? You might have Facebook's own tracking pixel, Google Analytics, and possibly another third-party software. Each source can provide information that the others don't track. Plus, any differences between the datasets could be informative (e.g. bot traffic that one source ignores while another source keeps).

Error Analysis (Post Modelling)

Possible next steps include collecting more data, splitting the problem apart, or engineering new features that address the errors. To use error analysis for feature engineering, you'll need to understand *why* your model missed its mark.

- **Start with larger errors:** Error analysis is typically a manual process. You won't have time to scrutinize every observation. We recommend starting with those that had higher error scores. Look for patterns that you can formalize into new features.
- **Segment by classes:** Another technique is to segment your observations and compare the average error within each segment. You can try creating indicator variables for the segments with the highest errors.
- **Unsupervised clustering:** If you have trouble spotting patterns, you can run an unsupervised clustering algorithm on the misclassified observations. We don't recommend blindly using those clusters as a new feature, but they can make it easier to spot patterns. Remember, the goal is to understand *why* observations were misclassified.
- **Ask colleagues or domain experts:** This is a great complement to any of the other three techniques. Asking a domain expert is especially useful if you've identified a pattern of poor performance (e.g. through segmentations) but don't yet understand why.

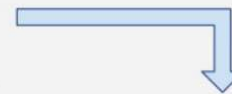
Example

Aggregating

Necessary when the entity to model is an aggregation from the provided data.

Original data (list of playbacks)

Content	Genre	Duration	Play Time	User	Device
Highway star	Rock	190	2015-05-12 16:29:33	User001	TV
Blues alive	Blues	281	2015-05-13 12:31:21	User005	Tablet
Lonely planet	Techno	332	2015-05-13 14:26:04	User003	TV
Dance, dance	Disco	312	2015-05-13 18:12:45	User001	Tablet
The wall	Reagge	218	2015-05-14 09:02:55	User002	Smartphone
Offside down	Techno	240	2015-05-14 11:26:32	User005	Tablet
The alchemist	Blues	418	2015-05-14 21:44:15	User003	TV
Bring me down	Classic	328	2015-05-15 06:59:56	User001	Tablet
The scarecrow	Rock	269	2015-05-15 12:37:05	User003	Smartphone



Aggregated data (list of users)

User	Num.Playbacks	Total Time	Pref.Device
User001	3	830	Tablet
User002	1	218	Smartphone
User003	3	1019	TV
User005	2	521	Tablet

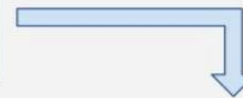
Example

Pivoting

Necessary when the entity to model is an aggregation from the provided data.

Original data

Content	Genre	Duration	Play Time	User	Device
Highway star	Rock	190	2015-05-12 16:29:33	User001	TV
Blues alive	Blues	281	2015-05-13 12:31:21	User005	Tablet
Lonely planet	Techno	332	2015-05-13 14:26:04	User003	TV
Dance, dance	Disco	312	2015-05-13 18:12:45	User001	Tablet
The wall	Reagge	218	2015-05-14 09:02:55	User002	Smartphone
Offside down	Techno	240	2015-05-14 11:26:32	User005	Tablet
The alchemist	Blues	418	2015-05-14 21:44:15	User003	TV
Bring me down	Classic	328	2015-05-15 06:59:56	User001	Tablet
The scarecrow	Rock	269	2015-05-15 12:37:05	User003	Smartphone



Aggregated data with pivoted columns

User	Num.Playback	Total Time	Pref.Device	# playbacks by device			Play duration by device		
				NP_TV	NP_Tablet	NP_Smartphone	TT_TV	TT_Tablet	TT_Smartphone
User001	3	830	Tablet	1	2	0	190	640	0
User002	1	218	Smartphone	0	0	1	0	0	218
User003	3	1019	TV	2	0	1	750	0	269
User005	2	521	Tablet	0	2	0	0	521	0

Example

- Simple linear models use a linear combination of the individual input features, x_1, x_2, \dots, x_n to predict the outcome y .

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

- An easy way to increase the complexity of the linear model is to create feature combinations (nonlinear features).

Example (House Pricing Prediction) Area (m2) # Rooms

Degree 2 interaction features for vector $X = (x_1, x_2)$

Price $\rightarrow y = w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$

Example

Date	No. of footfalls	Is Holiday?
2-Jul-17	12453	Yes
3-Jul-17	6543	No
4-Jul-17	1233	No
5-Jul-17	6000	No
6-Jul-17	4656	No
7-Jul-17	9800	No
8-Jul-17	14654	Yes
9-Jul-17	11234	Yes
10-Jul-17	7654	No

Item_ID	Item_Weight	Item_Price	Price_per_Weight
FDA15	9.3	249.81	26.86
DRC01	5.9	48.27	8.15
FDN15	17.5	141.62	8.09
FDX07	19.2	182.10	9.48

client_id	joined	income	credit_score	join_month	log_income
46109	2002-04-16	172677	527	4	12.059178
49545	2007-11-14	104564	770	11	11.557555
41480	2013-03-11	122607	585	3	11.716739
46180	2001-11-06	43851	562	11	10.688553
25707	2006-10-06	211422	621	10	12.261611

document_id	topic_id	confidence	ROUND(confidence*10)
25792	1205	0.9594	10
15454	1545	0.1254	1
78764	958	0.1854	2
21548	1510	0.5454	5
48877	25	0.3655	4

Example

	Attack	Defense	Attack^2	Attack x Defense	Defense^2
0	49.0	49.0	2401.0	2401.0	2401.0
1	62.0	63.0	3844.0	3906.0	3969.0
2	82.0	83.0	6724.0	6806.0	6889.0
3	100.0	123.0	10000.0	12300.0	15129.0
4	52.0	43.0	2704.0	2236.0	1849.0

Numeric features with their interactions

Flight Status Prediction

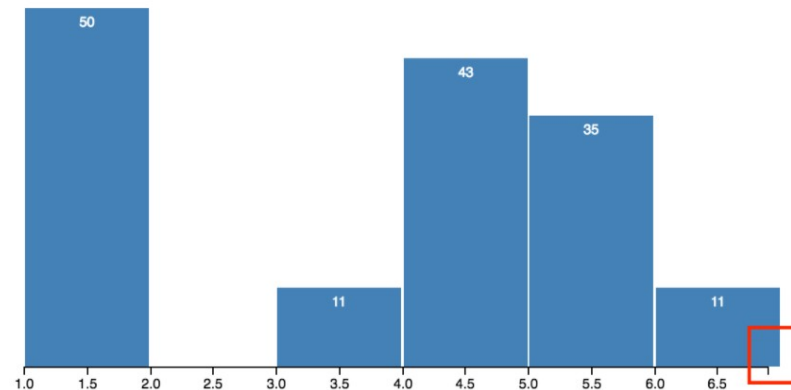
Date_Time_Combined			Status	Hour_Of_Day			Status
0	2018-02-14	20:40	Delayed	0	20	Delayed	
1	2018-02-15	10:30	On Time	1	10	On Time	
2	2018-02-14	07:40	On Time	2	7	On Time	
3	2018-02-15	18:10	Delayed	3	18	Delayed	
4	2018-02-14	10:20	On Time	4	10	On Time	

- Binning, also known as quantization is used for transforming continuous numeric features into discrete ones (categories). These discrete values or numbers can be thought of as categories or bins into which the raw, continuous numeric values are binned or grouped into.
- Each bin represents a specific degree of intensity and hence a specific range of continuous numeric values fall into it. Specific strategies of binning data include fixed-width and adaptive binning.



Binning

- **Fixed-Width Binning**
- Just like the name indicates, in fixed-width binning, we have specific fixed widths for each of the bins which are usually pre-defined by the user analyzing the data. Each bin has a pre-fixed range of values which should be assigned to that bin on the basis of some domain knowledge, rules or constraints.
- Binning based on rounding is one of the ways, where you can use the rounding operation which we discussed earlier to bin raw values.



Age Range : Bin

```
-----
0 - 15 : 1
16 - 30 : 2
31 - 45 : 3
46 - 60 : 4
61 - 75 : 5
75 - 100 : 6
```

	ID.x	Age	Age_bin_round
1071	6a02aa4618c99fdb3e24de522a099431	17.0	1.0
1072	f0e5e47278c5f248fe861c5f7214c07a	38.0	3.0
1073	6e14f6d0779b7e424fa3fdd9e4bd3bf9	21.0	2.0
1074	c2654c07dc929cdf3dad4d1aec4ffbb3	53.0	5.0
1075	f07449fc9339b2e57703ec7886232523	35.0	3.0

Binning by rounding

	ID.x	Age	Age_bin_round	Age_bin_custom_range	Age_bin_custom_label
1071	6a02aa4618c99fdb3e24de522a099431	17.0	1.0	(15, 30]	2
1072	f0e5e47278c5f248fe861c5f7214c07a	38.0	3.0	(30, 45]	3
1073	6e14f6d0779b7e424fa3fdd9e4bd3bf9	21.0	2.0	(15, 30]	2
1074	c2654c07dc929cdf3dad4d1aec4ffbb3	53.0	5.0	(45, 60]	4
1075	f07449fc9339b2e57703ec7886232523	35.0	3.0	(30, 45]	3

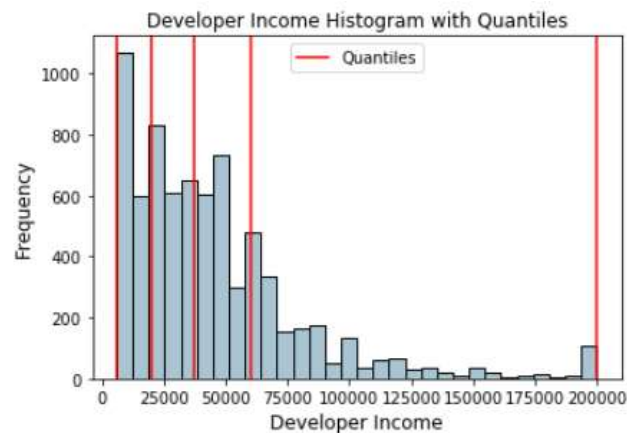
Custom binning scheme for developer ages

Adaptive Binning

- The drawback in using fixed-width binning is that due to us manually deciding the bin ranges, we can end up with irregular bins which are not uniform based on the number of data points or values which fall in each bin.
- Some of the bins might be densely populated and some of them might be sparsely populated or even empty! Adaptive binning is a safer strategy in these scenarios where we let the data speak for itself! That's right, we use the data distribution itself to decide our bin ranges

Adaptive Binning

- Quantile based binning is a good strategy to use for adaptive binning. Quantiles are specific values or cut-points which help in partitioning the continuous valued distribution of a specific numeric field into discrete contiguous bins or intervals.
- Thus, *q-Quantiles* help in partitioning a numeric attribute into *q* equal partitions. Popular examples of quantiles include the *2-Quantile* known as the *median* which divides the data distribution into two equal bins, *4-Quantiles* known as the *quartiles* which divide the data into 4 equal bins and *10-Quantiles* also known as the *deciles* which create 10 equal width bins



Histogram depicting developer income distribution with quartile values

```
quantile_list = [0, .25, .5, .75, 1.]
quantiles = fcc_survey_df['Income'].quantile(quantile_list)
quantiles
```

Output

```
-----
0.00      6000.0
0.25     20000.0
0.50     37000.0
0.75     60000.0
1.00    200000.0
Name: Income, dtype: float64
```

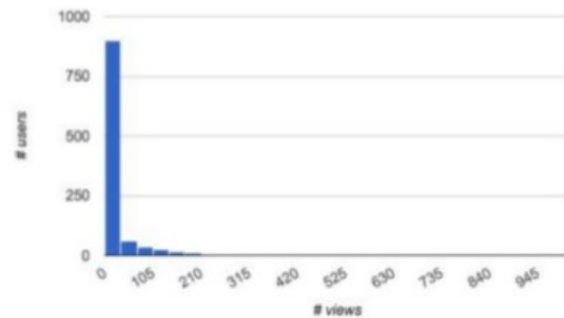
The red lines in the distribution above depict the quartile values and our potential bins. Let's now leverage this knowledge to build our quartile based binning scheme.

	ID.x	Age	Income	Income_quantile_range	Income_quantile_label
4	9368291c93d5d5f5c8cdb1a575e18bec	20.0	6000.0	(5999.999, 20000.0]	0-25Q
5	dd0e77eab9270e4b67c19b0d6bbf621b	34.0	40000.0	(37000.0, 60000.0]	50-75Q
6	7599c0aa0419b59fd11ffede98a3665d	23.0	32000.0	(20000.0, 37000.0]	25-50Q
7	6dff182db452487f07a47596f314bddc	35.0	40000.0	(37000.0, 60000.0]	50-75Q
8	9dc233f8ed1c6eb2432672ab4bb39249	33.0	80000.0	(60000.0, 200000.0]	75-100Q

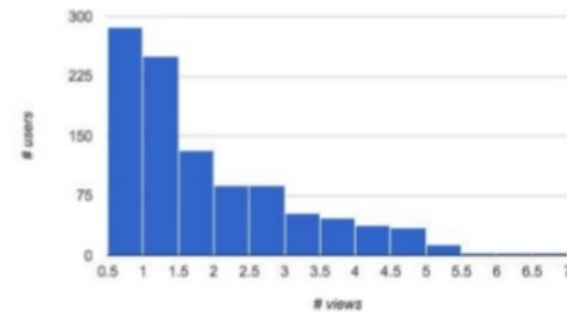
Binning

Log Transform

Smoothing long-tailed data with log



Histogram of # views by user



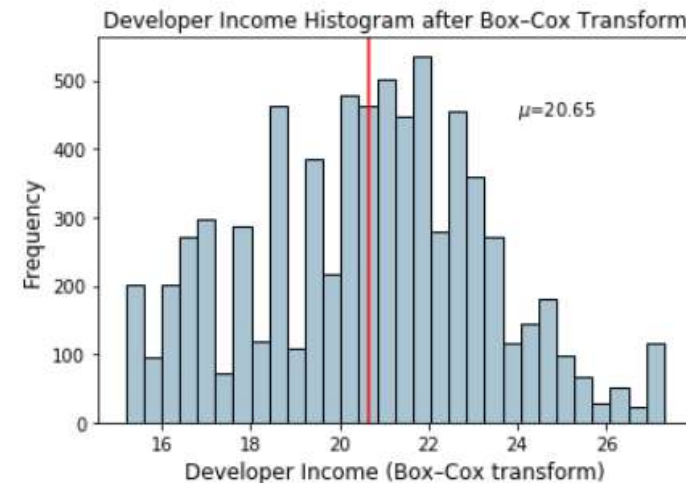
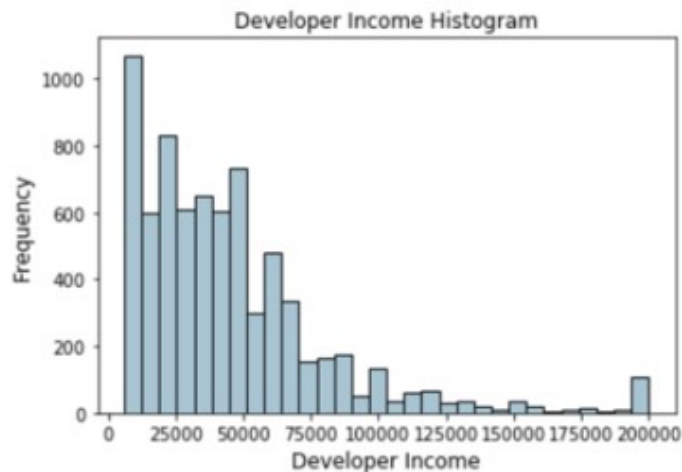
Histogram of # views by user
smoothed by $\log(1+x)$

Box-Cox Transform

A Box Cox Transformation is a simple calculation that may help your data set follow a normal distribution

$$x(\lambda) = \frac{(x^\lambda - 1)}{\lambda} \text{ for } \lambda \neq 0$$

$$x(\lambda) = \ln(x) \text{ for } \lambda = 0$$



Histogram depicting developer income distribution after Box-Cox transform



THANK YOU

