PRELAB #2

FOCUS: analyzing and organizing raw data & preparing it for the next stage of ML process. (modeling)

DATA MATRIX

- Data can come in many different formats.
- Data generally isn't stored in a format that's compatible for most ML models.
- MODEL DEVELOPER:
  - ↳ Transform the data appropriately
  - ↳ Prepare data for model performance where prediction, accuracy & generalization are your goals.
- Two mathematical constructs that represent data:
  1. Vectors = 1D array (EX. commonly used in Numpy)
  2. Matricies = 2D array (EX. set of N same size array stacked on top of each other.)

EX) Here displays a typical Pandas DataFrame. It has N rows & K columns where each row & column is a unique array. In our work we'll perform operations on both the column & rows arrays.

NOTE: only in SUPERVISED LEARNING.

Features | Label

Merchant Distance	Transaction Amount	Merchant Code (Type)	Is Fraud
5	39.01	1	0
2	112.81	1	1
8	4.99	1	0
6	1115.67	1	1
0.5	97.96	4	1
10.11	100	3	0
4	1.15	1	0
29.79	5.87	1	0
45.47	15.93	2	1
4	2500	1	0
0.95	25.66	2	0
21.33	2	5	0

Examples — Each row represents individual entities/units of the analysis.

Ex. when we do predictions on a trained model, we'll do them on a ROW ARRAY.

Ex. When we do data preparation work we often do operations on individual COLUMN ARRAY

- Defining what makes your tables' examples should be done in the early problem formulation stage.
- Your data preparation should then be centered on finding the appropriate sample of these units & defining the attributes associated w/ these units.
- With the units defined we will then define the columns [FEATURES]
  - ↳ The process of defining & coding up the features = FEATURE ENGINEERING
- LABEL is the item we're trying to predict.
  - ↳ During SUPERVISED LEARNING our goal is to ensure we have consistent set of FEATURES & LABEL for each EXAMPLE.

What we should consider when modeling: how many features & records are sufficient?

- ↳ GENERAL RULE: more is better of each.
- ↳ RIGHT ANSWER: Perform empirical evaluations.

WHO/WHAT ARE WE MODELING?

- Define your UNIT ANALYSIS in your INITIAL PROBLEM STATEMENT.
- The UNIT ANALYSIS needs to be considered in ALL STEPS in the MODEL BUILDING PROCESS.

EX) PROBLEM STATEMENT: FRAUD DETECTION: What's the likelihood that this transaction is fraud?
- ↳ MARKETING: Will this reader click on the ad I'm showing?
- ↳ RECOMMENDATIONS: What are the best tweets to display to this Twitter user?
- ↳ TELEMETRY: What's the likelihood that this engine will fail in the next 24 hrs?

Each can be solved with binary classification model & each has different unit of analysis that's specified in the PROBLEM STATEMENT.

- The data matrix will build to train a ML model will contain SAMPLE of these UNITS.
- Each SAMPLE are called EXAMPLES.
- The Database we'll be working with will probably have special ID columns that correspond to your UNIT OF ANALYSIS.
  - ↳ All of our DF/tables should include this ID/set of IDs.
  - ↳ These IDs are useful when we want to merge/join individual DF together & they'll be used as JOINT KEYS.

REMEMBER | ID COLUMNS ≠ FEATURES

SAMPLING TECHNIQUES

SAMPLING: The process of extracting subsets of examples from available universe & data.

↳ CONCERNS:

1. The population we want to manage
2. Managing the number of examples we'll use to model

↳ DEFINING SAMPLE POPULATION:

- More descriptive, though, than just the unit.
- Are there any attributes of those units that should be considered when sampling them from larger pool of units.
  - ↳ This should be specified during PROBLEM FORMULATION

UNIT ANALYSIS = ▩ SAMPLING = ▩

EX) We'll add extra attribute that'll serve as filter on full set of units.
These attributes should be tied to the core problem you're trying to solve.

PROBLEM STATEMENT: FRAUD DETECTION: What's the likelihood that this transaction is fraud?
- ↳ MARKETING: Will this reader click on the ad I'm showing?
- ↳ RECOMMENDATIONS: What are the best tweets to display to this Twitter user?
- ↳ TELEMETRY: What's the likelihood that this engine will fail in the next 24 hrs?

⇓ INSTEAD OF

↳ FRAUD DETECTION: What's the likelihood of ALL transactions being fraud?

⇓ With sampling

↳ FRAUD DETECTION: What's the likelihood that a web-based transaction is fraud?
↳ MARKETING: Will this non-subscribing reader click on the ad I'm showing? ⟹ subscriptions are FALSE
↳ RECOMMENDATION: What are the best tweets to display to this new Twitter user?

- Oftentimes you don't want your model to apply to every person in every situation.
- Sometimes you can have different models for different segments of the population.
- Your PROBLEM STATEMENT may dictate using different sampling populations.
- Build a model that's specific to each of these populations.
- WHY BE PRECISE W/ SAMPLE POPULATION? ⟹ GOAL: Generalization
  - ↳ A model trained on a given sample whose population is defined by having certain attributes may not generalize & perform well on a sample with different attributes.

IID (independent & identically distributed) SAMPLES: When we expect 2 independent samples to have the same distribution of features.

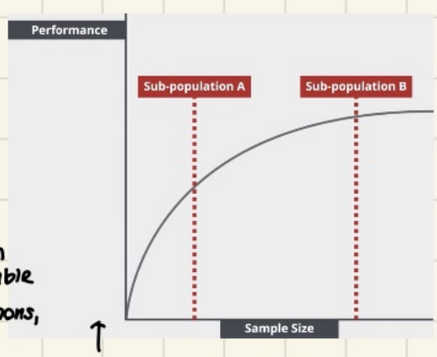
↳ We can achieve this by using a RANDOM NUMBER GENERATOR to select examples from a population.

SAMPLING STEPS:

1. Define the population of interests, including selecting the unit & key attributes to filter on.
2. Random selection from that population, where the size is determined by what's available and/or what's necessary for strong generalization performance.

GENERAL THUMB RULE: more examples lead to better models

HOWEVER: there may be diminishing returns ∴ we want sample size vs. performance to look like:



We may want to deliberately invest in more examples from subpopulation A to lift performance. Then it would be comparable to Group B.

Always aim to build models that treat people in different protected classes equally. It's possible that be certain classes are underrepresented in the data, the models are inaccurate for those particular group. If data is available then evaluate the model on different groups to test for this. If the model underperforms for smaller subpopulations, one trustworthy way to solve this is to resample the training data so that each group is equally represented.

DATA PREPARATION TECHNIQUES

In real life data is not always suitable for training out-of-the-box. There can be missing values, bad formatting, outliers, and data redundancy are some common problems a ML engineer has to overcome.

Below is an example dataset that contains some of the issues we just discussed.

Sample dataset

Name	Income	Credit Score	Occupation	Job Sector	Loan Status
John Doe	$76,000	650	Engineer	Engineering	Good
Gill Bates	$85,000	760	Nurse	Healthcare	Defaulted
Jane Doe	"95000.00"	0	Banker	Financial	Good
John Doe	$76,000	650	Engineer	Engineering	Good
Melon Usk		810	Flight Attendant	Transportation	Excellent
Barren Wuffet	5000/mo	35000	Contractor	Construction	Defaulted

① The entry for John Doe is repeated
② Income format is not standardized
③ Credit score contains 2 outliers: 0 & 35,000 likely to be errors as typical range is between 350 to 850.
④ The entry for Melon Usk contains a missing value.
⑤ Occupation & Job section are somewhat redundant as they tell similar stories.

KEY POINTS:

- Essential that raw data go through pre-processing steps such as cleaning, featuring engineering, and outlier handling.
- Data matrix is the ideal input format for ML.
  - ↳ The 1st N-1 column [NOTE: N = size] in a data matrix are features & the last column is the label.
  - ↳ Each row in a data matrix is a data point.
- EDA (Exploratory Data Analysis) helps engineers and scientists understand the basic properties of a dataset including its shape, statistical properties, and whether it contains outliers or not.

DATA UNDERSTANDING:

DEFINE THE PROBLEM: This is step zero of any ML project. Before we perform any action, we need to first answer the who and what we are modeling. This can be answered by defining the problem statement & breaking it down into units of analysis.

EXPLORATORY DATA ANALYSIS: EDA is the act of "poking around" the data for any obvious insights. This entails bringing in the data (or a subset thereof) into our coding environment, performing descriptive summary using common statistical libraries & making plots such as scatterplots and histogram for any obvious insight.

VISUALIZATION: Helps us understand our data better. Helps us determine the skew of data, shape of its distribution, and if any outlier is present.

DATA PREPARATION:

SAMPLING: Once we have a concrete idea of the problem we're trying to solve, we need to ensure our data is meaningful in the context of the problem. This can be thought of as having the data drawn from the same environment as production.

We want to ensure that our data points are independentially & identically distributed (IID), meaning that they should be drawn from the same distribution and are independent from each other.

SPECIFYING LABEL: The training dataset for supervised learning must have some "ground truth" (label) associated with each data point. The label is what we're trying to predict & it's the goal of our ML model.

THE DATA MATRIX: The underlying code for most ML models is optimized to process data in an N-dimension table AKA. data matrix, where each row is example/data point, the last column is label, and each remaining columns is features.

DATA CLEANING: Most data is inherently "dirty". This can be caused by problems such as system error, data corruption, and poor quality control. In data cleansing, we ensure our data does not contain missing values or unwanted outliers. When these conditions are encountered, we perform actions s.a. dropping the row or winsorization in order to ensure our data matrix is clean & free of data that are non-representative of the problem we're trying to solve.

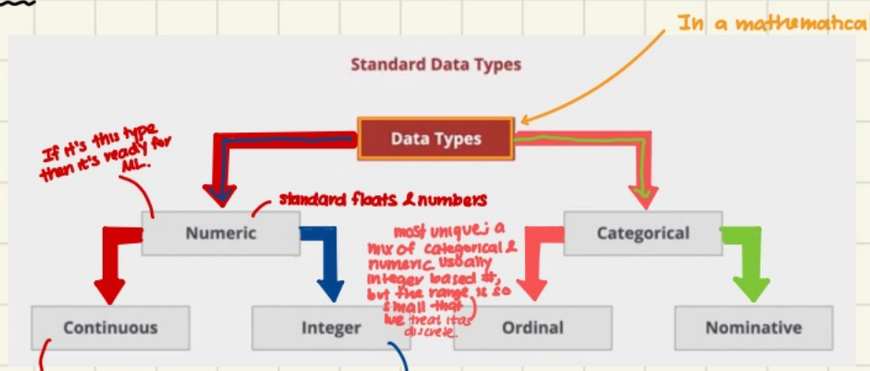
MODELING:

FEATURE ENGINEERING: The data available to us may not always be readily suitable for training a ML model. Sometimes this is due to formatting incompatibility, other times the models themselves are optimized for certain types of input. Feature engineering concerns with bringing data into the right format & representations required by the intended ML model.

CREATE LABELS & FEATURES

Continue our discussion on data preparation and focus on the columns of our matrix.

Different common DATATYPES:



- All ML methods ultimately operate on numbers so you can think of data prep as simply as creating a matrix full of numbers.
- Different datatypes have implications on how you might prepare FEATURES and how you can use them as LABELS for supervised learning tasks.
- EX) categorical — ordinal : surveys where people use 5 or 10 rating scale to express an opinion about something.
  - ↳ ORDINAL types are numbers ∴ can be used directly as FEATURES.
  - ↳ Best for ordinal features to be converted to binary indicator features
  - ↳ PREDICTING ORDINAL: use both regression & classification.
    - regression ⇒ You may get values that don't make sense
    - classification ⇒ can help you avoid this problem.

ONE HOT CODING

- CATEGORICAL — NOMINATIVE
  - ↳ A categorical variable → often represented as strings. ⟹ Most ML algorithm can't operate on strings. ∴ CONVERT TO NUMERIC
    → Database systems often encode discrete categories as integers. ⟹ DON'T TREAT THEM AS PURE #s
  - ↳ Nominative datatypes are typical basis for classification.
    - ↳ When used as LABELS we call the individual values CLASSES.
    - ↳ our model either directly predicts the CLASS or produces a score that represents the likelihood of belonging to a specific class.

SPECIFYING A LABEL

When building our data matrix for ML applications, we invest time in creating columns. Most columns represents FEATURES in process called FEATURE ENGINEERING.

In SUPERVISED LEARNING we first define the LABEL which nearly every aspect of the development process is centered around the label.

EASIER CASES:
- → There's a clear set of discrete choices to model
- → Each example maps cleanly to a single class
- → The label is directly observable

HARDER CASES:
- → The problem goal is subjective
- → There's multiple relevant labels to consider
- → The ideal label is harder to work with.
- → We don't directly observe the label we want to predict.

REMEMBER: Defining LABEL is tightly coupled with PROBLEM STATEMENT.

EX) Labeling for social network feed recommendations

PROBLEM GOAL: to maximize long term user satisfaction ⟹ SUBJECTIVE + HARD TO MEASURE AT SCALE ∴ we rely on PROXIES.
↳ This is what keeps the social network relevant & entertaining ∴ how we keep our customers

⇓ Primary proxy

User retention after 90 days ⟹ DIFFICULT TO WORK WITH + LONG TIME TO MEASURE + NOT HELPFUL MAKING SHORT-TERM DECISIONS

⇓ Proxy of the proxy

We can choose between DWELL TIME, LIKES, COMMENTS, etc. Various short term engagement

REMEMBER: NO right answer on how to label a problem

LABELING TRICKS TO SIMPLIFY THE PROBLEM : Take harder problem & simplifying so it's easier to work with.

① Grouping ordinal labels into high/low:

EX) Predicting ordinal outcome like an online review from 1-5. We can care about high vs. low ratings.
Map the ratings to a high vs. low scale & use BINARY CLASSIFICATION.

↳ rating = 4 OR 5, label = 1
↳ rating = otherwise, label = 0

There can be SUBJECTIVITY when defining what's high/low but core idea is SIMPLIFYING the problem.

② Converting time-to-event predictions to classification tasks.

EX) Predicting the time something might happen.
To approach the problem: convert to BINARY CLASSIFICATION

↳ whether a customer will still be a customer after some pre-specified # of days
↳ we'll lose some GRANULARITY in our predictions but problem will be easier to solve.

REMEMBER: changing label = changing problem statement.

REMEMBER: subjectivity is an inevitable part in design decisions ∴ always solicit feedback to ensure you're still align with the spirit of the OG problem.

pandas.Categorical(): Represents a categorical variable in classic R/S-plus fashion.

```
pandas.Categorical(values, categories=None, ordered=None, dtype=None, fastpath=False, copy=True)
```

- values: list-like — The values of the categorical.
- categories: index-like [OPTIONAL] — The unique categories for this categorical.
- ordered: bool; DEFAULT=False — Whether this categorical is treated as a ordered categorical. TRUE = resulting categorical will be ordered
- fastpath: OPTIONAL
- copy: OPTIONAL

pandas.get_dummies(): converts categorical variable into dummy/indicator variables

```
pandas.get_dummies(data, prefix=None, prefix_sep="_", dummy_na=False, columns=None, sparse=False,
                   drop_first=False, dtype=None)
```

- data: array-like, series or DF — Data in which to get dummy indicators
- prefix: str, list of str, dict of str, DEFAULT=None — String to append DF column names
- prefix_sep: str — If appending prefix
- dummy_na: bool, DEFAULT=False — Add column to indicate NaNs
- columns: list-like — Column names in the DF to be encoded

INTRODUCTION TO FEATURE ENGINEERING

GOAL: prepare our data for ML model to train on.

- Not much concerned about cleanliness of the data (Ex. outliers, missing data)
- FOCUS: mapping the appropriate predictive/causal concepts into a data representation & transforming it into a format that can be easily consumed by intended ML model.

Essentially we're answering: HOW DO WE GET FROM SOME DATA SOURCE TO A COLUMN IN A DATA MATRIX?

↳ 2 strategies: ① Select the right data to be our features.

↳ STEP 1: Selection, filtering, and fetching of the desired data from some data source into our ML environment.

↳ Machine's condition = FEATURE
↳ Operational status = LABEL

↳ STEP 2: Bring data into our ML environment.

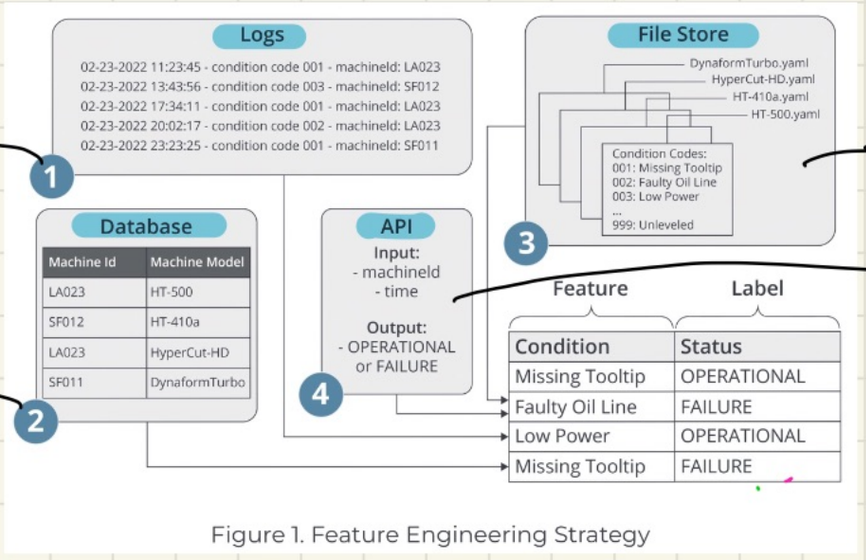


Figure 1. Feature Engineering Strategy

- We can repeat this step to bring in other features for constructing our data matrix.

② FEATURE TRANSFORMATION: Transform the data into a format that's suitable for the intended ML model.

↳ The types of feature transformation to apply depends on the model.

↳ common feature transformation techniques: BINARY INDICATOR, ONE HOT ENCODING, FUNCTIONAL TRANSFORMATION

PROCESS OF FEATURE ENGINEERING

2 STRATEGIES:

① Mapping predictive or causal concepts to data representation

↳ Aggregate various data assets into a single modeling matrix

↳ Converting data with different formats & locations into a common structure.

↳ create conceptual model of problem to help guide you.

> NOTE: performance usually comes from 1st strategy.
> If your not capturing the right concepts in your data ⇒ no algorithm is going to make a good prediction.

② Manipulating data so that's appropriate for common ML APIs.

↳ 4 main techniques: 1) Work converting log or database information into concepts you think would make for good features.

2) ONE HOT ENCODING

↳ categorical values absolutely needs to be converted to numeric form before it can be used as a feature

↳ Not optional when you have categorical or text or string data.

DOMAIN-DRIVEN FEATURE ENGINEERING

Cases where feature engineering isn't important:

EXAMPLE 1 IMAGE RECOGNITION

EXAMPLE 2 LANGUAGE TRANSLATIONS

Both involve DEEP NEURAL NETWORKS where they have the property of they can automatically learn feature concepts from RAW DATA.

PRINCIPLES: when we map concepts to data representations

① THINK LIKE A SCIENTIST: Based on your knowledge of the event you're trying to model, identify likely causal factors & test them empirically.

- The best features in the model are believed to cause the outcome.
- Don't limit yourself to proven causes.

② SEEK THE WISDOM OF DOMAIN EXPERTS: Interview domain experts to gain insight into hypothesized or known causal factors.

- You can understand what features NOT to build ⟶ legal + system constraints?
- Knowledge on how to approach the problem

③ PRIORITIZE & ITERATE: Plan your feature engineering tasks in advance & prioritize those with highest expected likelihood to be predictive.

- use your collective intuition & knowledge of the domain & plan ahead.
- create a list of all the features you plan to build & seek input on their expected value.
- Then build the features & evaluate them as you go along.

EX) TWITTER RECOMMENDATIONS ON WHO TO FOLLOW

- LABEL: follow/not follow after being recommended.
- we're creating concepts that either cause/highly correlated with the outcome of interest.

EXAMPLE CONCEPTS FOR THE FEATURE ENGINEER PROCESS:

Why do I follow someone? ← Think about your own experience; what thought process do you have when you decide to follow someone.

⊗ ↳ Do I find the person interesting? ← we want to AVOID abstract & subjective concepts.
We want to define factors that are specific enough that can be coded up in an unambiguous way.

✓ ↳ The person posts about topics that I'm interested in.
↳ Represent it as a match between topics that they post & topics I explicitly followed.

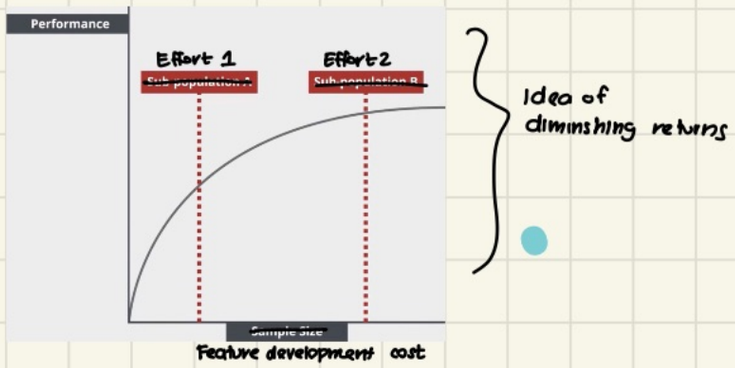
✓ ↳ This person follows me.
↳ Assume that I'd reciprocate.

✓ ↳ People I follow also follow this person.
↳ Friends I consider interesting also consider someone else interesting. I would as well.

✓ ↳ The person is very popular.
↳ other people find interesting than I might as well.

- These examples meet our condition & we can translate those features that make it into our data matrix.
- If we plot feature development cost w/ model performance then:

↳ once we hit those diminishing returns, we can then evaluate how much additional effort do we really want to put into the feature engineering.



FEATURE TRANSFORMATIONS

Different ML models are intended & optimized for different inputs. This means that ML engineers need to transform the features into the appropriate format or values in order to train models properly.

ALL OF THE ALGORITHM REQUIRE ANY CATEGORICAL VALUES TO BE CONVERTED TO NUMERICAL FORMAT

↓

POSSIBLE TRANSFORMATION TECHNIQUES:

BINARY INDICATOR: In some cases, it makes sense to represent numeric or categorical data as binary values.

- BENEFIT: makes the data smaller == lower computing cost
  makes our final model simpler == desirable if we want our model to ultimately be more general than specific
- EX) Cast movie rating greater than or equal to 3 to Good & movie rating less than 3 to bad.
  EX) Cast glucose level of 100 or higher as Abnormal and below 100 as Normal.
  EX) Group countries into English Speaking or Non-English Speaking.

ONE HOT ENCODING: Popular method for categorical data. ML algorithms operate on numerical inputs ∴ we have to transform categorical data into some form of numerical representation. The idea is to convert K categories into an array of K binary values prior to being consumed by a ML model.

- EX) Feature called weather which has 5 distinct values: Sunny, Overcast, Rain, Snow
  2 other features: elevation & zip code ← BOTH have numerical types ∴ don't need to be one-hot encoded

To one-hot encode weather feature: → Replace our feature (column) weather w/ 4 new features (columns)
→ One new feature (column) for every distinct value in the weather column.
→ Each new feature would have a binary value: 1/0.
↳ 1 = every row in which it appeared in the OG weather column
↳ 0 = otherwise

Data before One-Hot Encoding

Elevation	Zip code	Weather
1000	92651	Sunny
40	12834	Overcast
500	85532	Rain
232	43645	Snow
157	23426	Rain

One-Hot Encoding

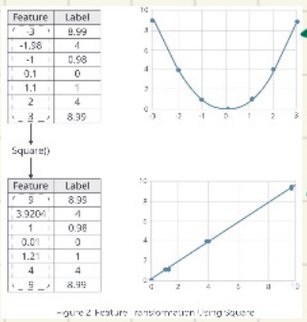
Data after One-Hot Encoding

Elevation	Zip code	Sunny	Overcast	Rain
1000	92651	1	0	0
40	12834	0	1	0
500	85532	0	0	1
232	43645	0	0	0
157	23426	0	0	1

Figure 1. One-Hot Encoding

FUNCTIONAL TRANSFORMATIONS: We apply a function to convert one numeric value to another based on some function.

- EX) Converting a feature to log scale or to the square.
- EX)



← our model more complex to capture the relationship among labels & features:
↳ more data to train
↳ slower in making predictions

transforming the feature values so that the relationship is linear
↳ simplify the model
↳ improve prediction speed.

Figure 2 Feature Transformation Using Square

INTERACTION TERMS: In some cases the combined effect of one or more features leads to a strong predictor than each feature alone. In such cases we can form additional feature by multiplying the individual features (or other mathematical operations) to arrive at a 3rd feature.

X1	X2
1	2
4	6
2	3

→

X1	X2	X1X2
1	2	2
4	6	24
2	3	6

Figure 3. New interaction terms created by multiplying X1 and X2

BINNING: Sometimes we want to convert numerical values into discrete bins when the ordinal nature of the numerical values themselves aren't necessarily indicative of that label.

- Binning is best visualized in HISTOGRAMS
- Useful in reducing the complexity of the model to achieve GENERALIZATION
- EX) Binning the weights of individuals into groups in an attempt to predict their risk of heart diseases.

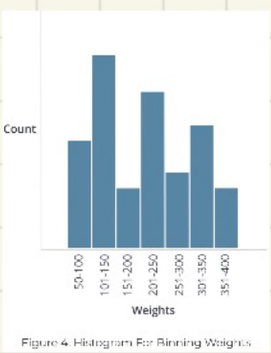


Figure 4. Histogram For Binning Weights

SCALING: Some models perform better when features are scaled to some standard range, while others have no problems accepting an input of a wide range.

- Most common types of scaling approaches:
  - ↳ STANDARD SCALER: transform the values of a feature to have a mean 0 & standard deviation of 1.
  - ↳ MIN-MAX SCALER / MIN-MAX NORMALIZATION: transform the values of a feature to a range between some min and some max value (oftentimes 0 & 1), while keeping their relative distance the same.