* **Analytics** **Modeling**: selecting and specifying the correct analytic model(s) to combine into a solution, compiling, building and/or forecasting the necessary data sets, + interpreting model output to make suggestions that match the organization’s needs, priorities, + structure
* Analytics can helps answer many types of questions:
* **Descriptive** 🡪 explanation of what happened
* How much effect a new carry-on luggage scanning system had on airport security wait times, when did a piece of manufacturing equipment dropped below acceptable level, which sets of customers are most alike in buying patterns + what factors are most important in determining customer similarity, is a new medical Tx better that a current one
* **Predictive** 🡪 what’s going to happen
* Global demand for crude oil next year, 5 years, or 10 years from now, Google’s stock price next year + the uncertainty in the estimate, who likely someone w/ certain characteristics will eat in a certain restaurant if they have a coupon
* **Prescriptive** 🡪 what action would be best?
* How should traffic lights be timed + synchronized to best reduce delays during major sporting events, when/where should oil tankers pick-up + make deliveries, strategies airlines could use to get passengers to destinations quicker before/during/after a snowstorm
* **More** **general**
* New tech installed to track new detailed info 🡪 how to monetize it + create value for our company
* **Modeling** – taking real-life situation + expressing the key parts of it mathematically so we can analyze the math + turn the mathematical analysis into real-life solutions/recommendations
* **Model =** actual mathematical expression of the real-life situation, but “model” could mean 3 slightly different things:
* Regression = a model
* More granular/level of detail 🡪 regression based on size, weight, distance = all this added detail is still a model
* More granular/level of detail 🡪 regression estimate 37 + 81\*Size + 76\*Weight + 4\*Distance on size, weight, distance = specific equation that is still a model

***Module 2 – Classification***

* Classification = putting things into categories.
* simplest examples: 2 categories (often just yes + no)
* bank differentiates between loan applications who will fully repay + those who won’t, security agency differentiates between citizen + potential terrorist, email filter differentiates between spam + real mail, legal document system differentiates between documents relevant or irrelevant to a specific case, is an organ safe to transplant or does it carry a deadly infectious disease.
* Examples w/ > 2 categories
* political consultant differentiates between supportive voters, opposition, + undecided voters, a paleontologist differentiates between many different species of dinosaurs to determine which a newfound bone belongs to.
* Each of these classification questions will require some data in order to get answers.
* Loan applicants 🡪 data on income, credit history, age, family size, assets, liabilities, + more.
* Based on attributes of previous loan recipients + observations of whether each loan was repaid or not, the bank can build a model to help classify future applicants.
* Can plot each past recipient's info on a graph w/ x = credit score + y = household income
* Ex: CS = 730 making $76K/year + if they repaid, mark in green, if not mark in red
* In real life there might be other factors, each of which would be a dimension 🡪 Length of time as a customer of this bank, # of dependents, total assets + liabilities, + so on.
* Can draw a *line* separating between green points + red points 🡪 to decide whether to offer a loan to a new applicant, see where the new applicant's data point is relative to the line.

*Choosing a Classifier*

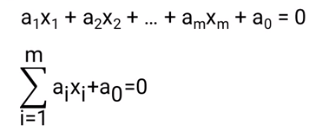
* In general, we want to choose classifier lines that are further from making mistakes
* But if this is impossible, we want a **soft classifier** which gives as good of a separation as possible rather than a **hard classifier**, which separates perfectly
* Want to figure out the trade-off between *near* and *actual* mistakes, depending on the importance in the situation (cost of giving a loan that won’t be repaid vs. cost of denying a good applicant, or cost of eating a poisonous plant vs. not eating an edible one)
* More costly a bad decision = more we want to move the line away from it
* Bad loan = 2x as bad as missing a good loan 🡪 move line towards green dots = less risk
* Only 1 attribute is need for classification (x-axis or y-axis value)

*Data Definitions*

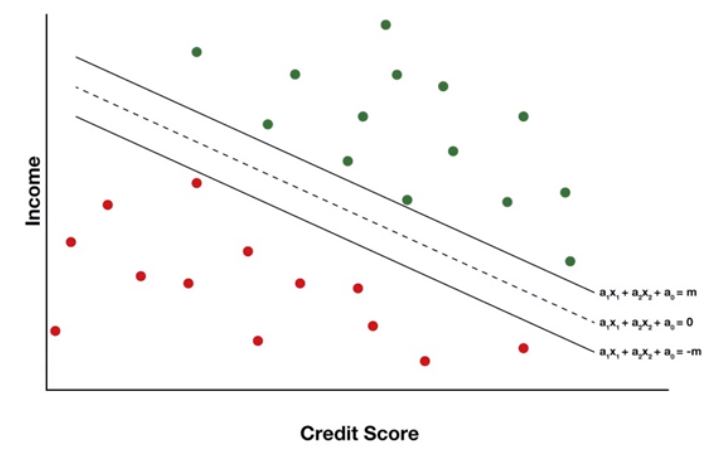
* Every row in data table = **data point** = *single observation* of info
* Every column in a data table = **attribute/feature/covariate/predictor,** but 1 **response**/**outcome**, the answer for each data (whether one repaid a loan, or how much % of the loan repaid, # of sales per on that day, etc.)
* Structured 🡪 can be described + stored in a structured way (letters and/or numbers, words, etc.)
* Categorical = Numbers have no magnitude (ZIP), or could be **binary** (only 2 values possible, sometimes treated as quantitative)
* M/F, hair color
* Quantitative 🡪 #’s that have meaning
* credit score, sales, age
* Unstructured 🡪 cannot be easily described + stored
* Written text, tweets, images, videos, music
* Sometimes data is unrelated (no relationship between records 🡪 different customers or applicants)
* Sometimes related via time = **time-series data** 🡪 same data recorded over time period (each day, year, minute, etc.) in equal intervals (typically)

*Support Vector Machines (SVM)*

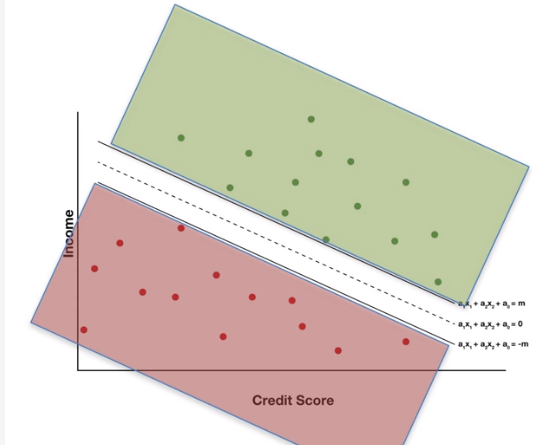
* Basic mathematical model for solving classification problems
* Ex: data set w/ n data points + m attributes, therefore x(ij) = ith attribute of jth data point, yj = response for jth data point
* x(1j) = credit score of person j y(2j) = income of person j yj = 1 if j is green, -1 if red
* The line to differentiate between red + green is defined by coefficients a(1)-a(m) + an intercept a(0)



* Parallel classification lines have the same coefficients, just different intercepts
* We can create 3 lines, 2 to separate the red + green points into their own sections, and 1 between them such that a(0) is the intercept of this middle line, our **classifier**

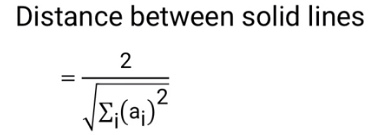


* Want an equation such that we classify points correctly + get the maximum gap/**margin** between the parallel separator lines
* To correctly classify all green points, we need our equation’s value y(j) to be >= 1, and for y(j) to be <= -1 to correctly classify all reds (using 1 and -1 for scaling, it could be any number)

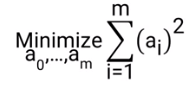
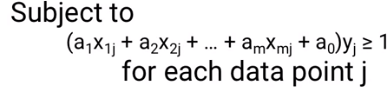


* Since both equations evaluate to y(j), we can combine these equations to get this:

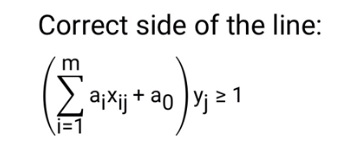
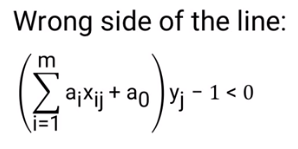




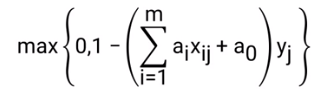
* From the above, we see we want to minimize the sum of the squared coefficients (excluding a(0)) to maximize the margin
* **Hard separation problem** = over all the a’s, minimize the sum of the squares of the a’s, subject to our combine equation being true for all data points

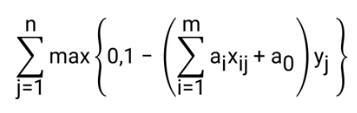
 

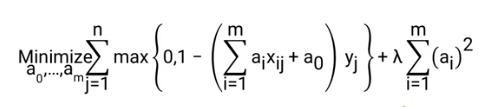
* So we can only choose values of a(i) that make this true in our minimization problem
* If it’s not possible to perfectly separate these points, we use a soft classifier + account for errors in classification + trade-off the errors made and maximizing the margin
* Can calculate classification error for each data point j

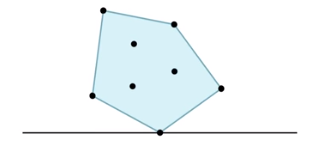
* How far on the wrong side of the line is the error
* Error for a data point j is whichever is larger, 0 or how far it is away the line on the wrong side



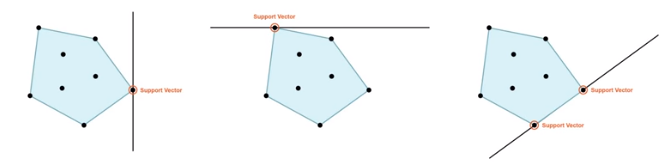
* Total Error 🡺 
* To trade-off between the maximized margin and minimized error, we can pick a value lambda to minimize the combination of error + margin



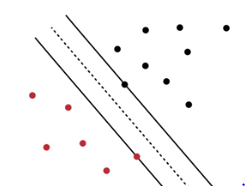
* As lambda gets larger, the second term gets larger, so the importance of a large margin outweighs the importance of a small total error in classifications, and vice versa.
* This is out **Support Vector Machine model**
* Imagine a set of points, and we connect the dots to create a border on the outside, the **convex hull** of the points, and use a straight line to “hold up” the resulting shape



* If the shape is correctly balanced on the line via this points, that touching point is the **support vector** (each points is a **vector** of data)
* We don’t need the line of the support vector to be *below* the shape, as from the mathematical definition a support vector could be supporting from the side or top
* Could even have more than 1 support vector, depending on the line we choose

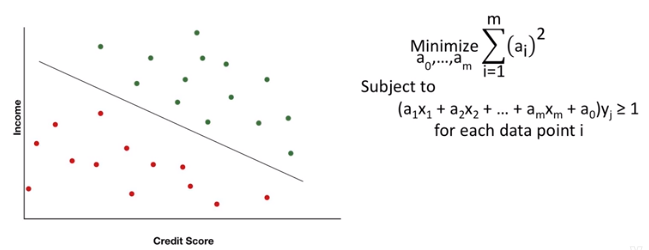


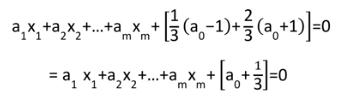
* In the **Basic Support Vector Machine Model for Classification =** looking for 2 parallel lines that are as far apart as possible while still having all points of each color on 1 side of the lines.



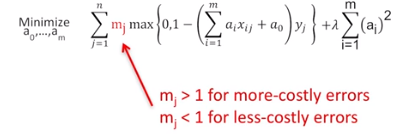
* See 2 points are support vectors
* B/c our model takes a data set + automatically determines where those 2 solid lines should be + what the support vectors are, it's called a Support Vector Machine Model.
* Remember the classifier we're looking for is actually *between* the 2 parallel lines + therefore between the support vectors, *NOT* touching *either* of them.
* The line the classifier returns is actually NOT 1 of the lines touching a support vector.

*Advanced SVM*

* Refresher 🡪 From a data set we can graph our points + use SVM to find a classifier w/ maximum separation/margin between the 2 sets of points = **hard classification**
* 
* If it’s impossible to avoid classification errors, use SVM to find a classifier that trades-off reducing errors + enlarging the margin, possibly moving the classifier if 1 error type is more costly than the other = **soft classifier**
* Ex: Perfect classification 🡪 classifier we choose can depend on the intercept value, a(0)
* Ex: Hard classification = Giving a bad loan = 2x as costly as withholding a good loan = choose intercept = 1/3\*(a(0) – 1) + 2/3\*(a(0) + 1)



* Ex: Soft classification = add an extra multiplier for each type of error w/ a larger penalty the less we want to mis-classify that type of point

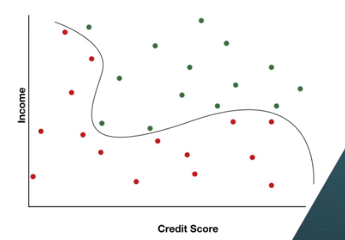




* It’s more costly to make classification errors for points 21-50 in the above (larger m(j))
* Remember to maximize the margin, we minimize the **sum of squared coefficients**
* This causes problems if data has very different scales (credit score [300-850] vs. income [> 1000 up to millions] 🡪 different by 2-3 **orders of magnitude** (2-3\*10 times bigger))
* When adding squares, a small change in one could swamp a *huge* change in the other
* Can fix by **scaling** the data to approximately the same order of magnitude before running SVM
* Some software does it automatically
* Once data is scaled, the coefficients a(1)-a(m) can be used to pick out attributes not necessary for classification
* Ex: almost vertical classifier:



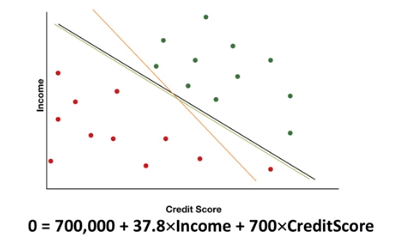
* Implies the vertical axis attribute’s value has almost no impact/is irrelevant to the classification
* Note this is only good for 2D graphs w/ 2 attributes
* With more attributes such that we can’t have a simple graph, we check coefficient values
* If close to 0, the attribute probably has negligible effect on the outcome variable + is probably not relevant for classification
* Note: SVM can work with a lot of attributes + the classifier does NTO have to be a straight line
* SVM can be generalized via **kernel methods** that allow for nonlinear classifiers (such as kernel SVM function in R)

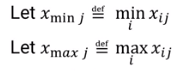


* If looking to answer classification w/ probabilities instead of straight “yes/no” answers, logistic regression would be better

*Scaling + Standardization*

* Ex Model: Given set of data {x(ij)}, where x(ij) is the jth factor of the ith data point, the classifier is the line 0 = a(0) + sum of a(j)\*x(j) over all factors in which we maximize the distance between classifier lines via minimizing the sum of squared factors a(j)^2
* Ex: 5x + 700x for income in 10’s of thousands and credit score 🡪 SSC = 5^2 + 700^2 = 490025.
* But if we change the credit score by just 1 🡪 5^2 + 700^2 = 491,426 🡪 increased SSC by 1401
* To change the income coefficient to get the same amount of change while keeping credit score at 700, we need 37.8^2 + 700^2 = 491426 🡪 a 600% change
* Data is at different scales = coefficients are on different scales = SSC is much more sensitive to changes in 1 coefficient than the other = SVM model won’t work well:



* Need to adjust the data so that the attributes are on the same scale via 2 methods
* **Scale** down to same interval *linearly*
* Common method is to scale data between 0-1
* Scale factor by factor 🡪 for each factor j: 
* Then for each point i, it’s new scaled value 🡪 **x(ij scaled) = [x(ij) – x(min j)] / [x(max j) – x(min j)]** 🡪 
* Scale between set range b and a 🡪 take the 0-1 scale + multiple by (a-b) and add b:



* **Standardization**: Scaling to normal distribution to measure how far from the mean each data point x(ij) is
* Most common way 🡪 scale to mean = 0 and SD = 1
* So mean mu(j) = **[sum( x(ij) ) from I = 1 to n] / n** and **SD = sigma(j)** and for each data point I we have **x(ij standardize) = x(ij) – u(j) / sigma(j)**

v  

* Which method to use:
* Some models need data w/in a bounded range = scaling
* Neural networks, optimization models needing bounded data, batting averages (.000-1.000, RGB color intensities (0-255), SAT scores (200-800)
* Other models work best w/ standardizations
* Principal component analysis, clustering
* In many cases it’s not clear which method is better 🡺 TRY BOTH
* Either way, it’s important to do one or the other on input data

*K-Nearest Neighbor (KNN) Classification Model*

* This is a simple model for solving classification models w/ more than 2 classes
* Ex: Dataset w/ 2 predictors + a response
* Instead of calculating a classifier line, we *assume each new applicant is similar to previous applicants that it’s data point is closest to*
* Ex: new data point is close to 4 green points + 1 red point 🡪 assume more likely to be green
* Can pick any # of n-closes points to estimate a class
* Process 🡪 Pick k closest points/nearest neighbors 🡪 new point’s class = most common among the k neighbors
* Some complexities:
* When looking for k closest points, there are multiple ways to measure distance
* Most common = straight-line 🡪 **Sqrt( Sum [ Abs[x(i) – j(i)]^2) over all points i-n ] )**



* Some attributes may be more important to the classification than the other
* Handle by weighing each dimension’s distance differently 🡪 more weight = more impact of the dimension’s distance

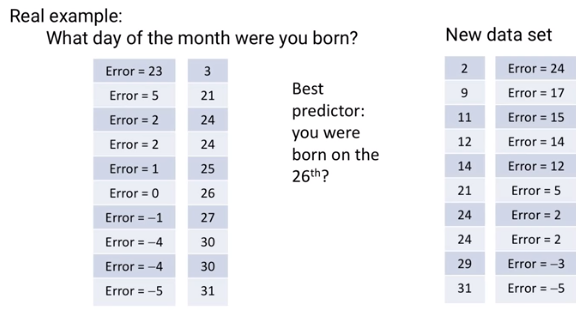
 🡪 finding this weight can be hard

* Extreme case 🡪 attributes that aren’t important at all in classification 🡪 ignore them/give them w(j) = 0
* Choosing good value of k 🡪 have to try different values + measure how well each one works via **validation** = measuring quality of a model

***Module 3 – Validation***

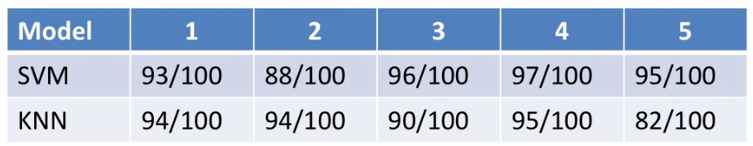
*Intro to Validation*

* Measuring how good a model is
* Ex: 24 data points, 3 incorrectly classified = 87.5% accuracy 🡪 WRONG
* This is fitting our models on the data we **trained** it on
* Too optimistic of an accuracy
* Any data set has 2 types of patterns:
* **Real effect** 🡪 real relationship between attributes + response
* **Random effect** 🡪 random but *looks* like a real effect
* Don’t know which patterns are real and which are random 🡪 when fitting to training set, we’re finding a model that fits both real AND random effects
* But when using model on new/different data, only real effects are still present + random effects will be different since our original performance measures the *training set’s own random effects*
* performance on test data won’t be as good 🡪 only matching real effects but not the random effects

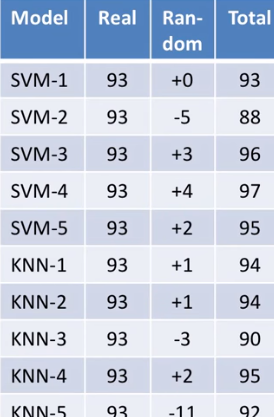


*Validation and Test Data Sets*

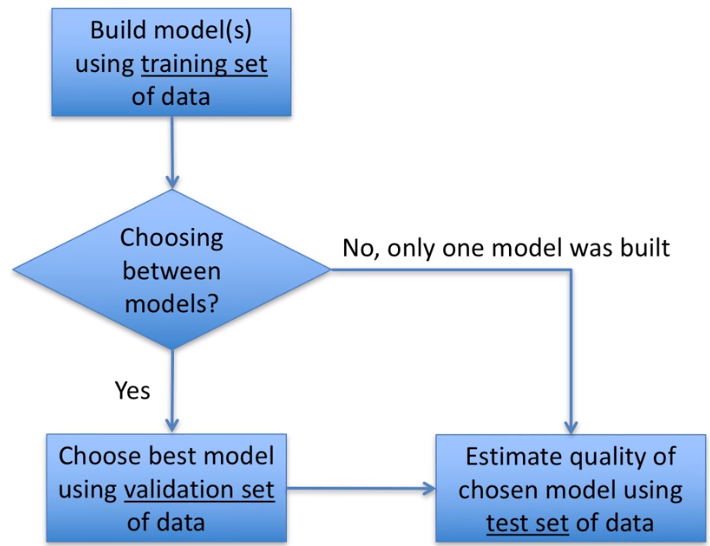
* 1 Way to measure model performance via 2 sets of data
* Use a larger set of data to fit the model 🡪 **training set**
* Use a smaller set of data to measure model’s effectiveness 🡪 **validation set**
* Ex: Loan applicants 🡪 create classifier on training set + measure effectiveness in validation set to see model is 90% accurate in training + 80% accurate in validation
* Most likely that 90% = too optimistic, 80% more accurate measure of effectiveness
* **Split** data 🡪 larger set to train/fit model + smaller validation set to estimate effectiveness of model
* What if comparing more than 1 model to use? Like 5 SVM models + 5 KNN models to choose the best of the 10?



* We’d measure effectiveness on validation set + pick the best performance 🡪 SVM 4
* BUT there’s a problem 🡪 even on the validation set, **observed performance** **(OP)** estimates include *some* random effects (OP = REAL quality + RANDOM effects)
* Sometimes randomness makes performance look worse than it really is, sometimes better
* So high-performance models are more likely to have *above-average random effects* 🡪 SVM 4 may have a lot of randomness
* So OP of a chosen model is probably too *optimistic*
* Ex: Each model performs equally 🡪 only difference therefore is due to randomness



* So quality on validation set may not be the true quality, and may be lower
* Once we’ve used the validation set to choose a model, we can’t necessarily use its performance as an estimate of model quality b/c it’s performance has a higher than average chance of being inflated by luck/randomness
* So to deal w/ this we do the same thing all over again 🡪 select model we want, use its performance on a 3rd set of data, **training set** to estimate the true performance of a model

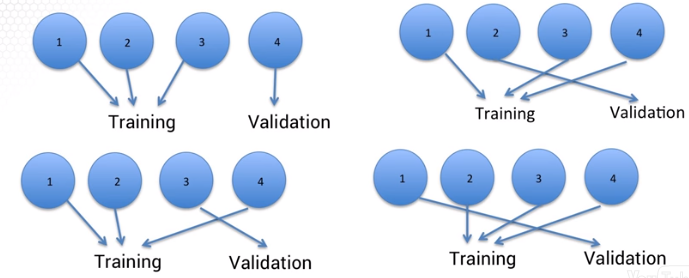


*Splitting Data*

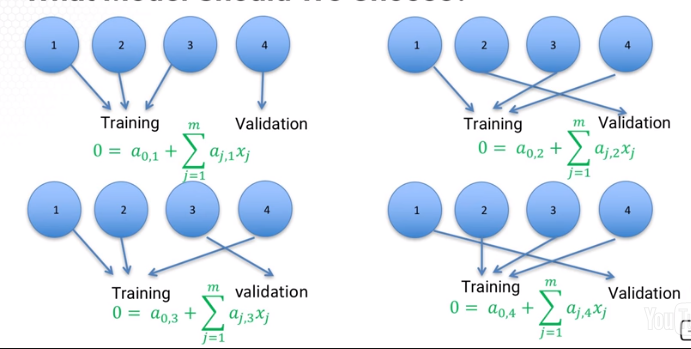
* No hard and fast rule for how much data should be in each set
* When working w/ 1 model 🡪 only need training (70-90%) and test (10-30%) sets
* Want more data in training so we have as much data as possible + can fit to as many important data points as possible
* Still want test set has enough data to cover important data points
* When working w/ 2+ models 🡪 need training (50-70%), validation, and test sets, w/ data being split equally between validation + test sets
* So, we want training to be LARGE and validation and test to be large ENOUGH
* How to split 1k data points, 60% training and 20% validation + test each
* 1) Random
* Randomly choose 600 points for training, then 200 out of the remaining 400 for validation, and the last 200 are used for test
* 2) Rotation
* Select point 🡪 move to next set 🡪 select point 🡪 next set and so on
* 5 points = training 🡪 validation 🡪 training 🡪 test 🡪 training
* Advantage of this method = can make sure each part of the data is equally separated
* Ex: data spans over 10 years
* Random approach may give 1 set too much early or late data, while rotation equally spaces them out
* Disadvantage of rotation = may introduce/create bias
* For DOW 🡪 each 5 point rotation puts all Mondays into 1 set, all Tuesdays into 1 set, etc.
* 3) Combined approach
* Randomly put 60% of Monday data for training, 60% of Tuesday data for training, etc.

*Cross-Validation*

* Now what if our data set only has couple of important points and they all show up in the validation or test set after splitting?
* This data isn’t in training, ,so the model doesn’t know about it + doesn’t train itself on these important points
* **Cross-validation** helps avoid this issue, through several types, such as **K-Folds Cross Validation**
* Ex: 20% of data in test, other 90% for training + validation in 1 step
* Split data into k = 4 parts
* Then do the training 3 times on the slightly different data sets and validate on the 4th slightly different data set, such as train on 1, 2, 3 and validate on 4
* THEN train on 1, 2, 4 and validate on 3, and so on



* So now every data point has been used to train 3 of these models, so we don’t have to worry about important data that may have been left out
* When comparing models to see which to choose, take the average of all 4 evaluations
* Generally, we’re splitting training and validation into k parts and for each of the k parts, training the model on all other parts and validation on the 1 remaining part, then average the k evaluations to measure/estimate model quality
* There’s no standard # for k, but k = 10 is common
* NOTE: Once we’ve used CV to choose a model, *what exactly is the model we’ve chosen?*
* w/ a k = 4-fold CV method on a SVM model, for each of the 4 ways to split we get a slightly different classifier



* We don’t use ANY of these classifiers, and do NOT average out the coefficients across the 4 splits (no guarantee this will work well)
* We instead, once selecting a model, we train it again, using all 4 parts of the data together
* K-Folds CV helps make better use of data and its averaging process helps give better estimates of model quality to help choose a model more effectively