PRELAB #8

ML engineers are accountable for the stability & performance of their model systems, where performance can be defined in multiple ways. There are several ways in which ML systems can cause problems.

- Factors to consider when preparing data & perform well in a production environment is: FAIRNESS & ETHICAL AI.
- Building effective ML models takes a lot of care: → Defining the problem
  → Preparing the data
  → Training & validating different model candidates

There are several places where the model can break down.

- Everything we do in this field requires us to consider common issues associated with SWE S.A. bugs in code, deployment problems, scalability, security features. On top of SW issues we have to consider failure modes that are specific to ML.

CAUSES OF FAILURES:

- Code errors
- Issues with API
- Issues with datatypes
- Silent failures

3 CLASSES OF MODEL FAILURE:

1) PERFORMANCE FAILURES
   - ↳ When evaluating on a test set your model may have poor performance or your model's performance might be too good to be true.

     OR

     May seem strange at first but can lead to overconfidence in a system
2) EXECUTION BOTTLENECKS
   - ↳ Data preparation or modeling process not terminating in a reasonable amount of time.
   - ↳ CAUSES: → misalignment between the data you have and the tools and hardware you're using to process the data.
3) SOCIETAL FAILURES
   - ↳ The ML model produces unintended discrimination or disparate impact and generally lacks accountability.
   - ↳ We use the phrase societal failure because this is where the failure exists between the owner of the modal and the societal context in which the model operates.

     It could be that models with societal failures perform well & generate a lot of value for the modal owner.

     As a result there are often misaligned incentives in remediating such failures.
   - ↳ Adopting methods to avoid societal failures which we'll call FAIR AI, being mindful of ML best practices early on in your model's development is one way you can contribute to reducing this type of failure mode.
   - ↳ Often CAUSED by POOR PLANNING.

GOOD PROBLEM DESIGN & PREPARATION

- Avoiding common ML failure mode usually starts with GOOD PROBLEM FORMATION
- As a MLE there are several subjective choices we'll have to make AKA. SUBJECTIVE CHOICES DESIGN DECISIONS (not empirical choices)
- Thoroughly understanding the constraints in which you operate & getting peer alignment ahead of doing any development is one way to proactively avoid the common ML failure modes.
- COMMON CONSTRAINTS:

DATA

1. Ask yourself whether you have the right data given specific problem statement
   - ↳ When considering whether the data is right for your application, you should look at whether your label captures the spirit of the problem statement, whether you have access to features that might predict the label, & whether you have access to a representative population.
2. Consider the data size
   - ↳ You can have TOO MUCH or TOO LITTLE data
   - ↳ Knowing this in advance can help you determine which modeling algorithms would be acceptable to test & what kind of computing systems you should set up.
3. Ask yourself, do you understand a particular algorithm you're planning to use?
   - ↳ Building a good model requires you to really understand the nuances of the underlying model, in particular, the art of managing the hyperparameters
4. Work through scalability requirements of your system.
   - ↳ Scalability requirements dictate what features & how many are reasonable to deploy & which modeling algorithms are appropriate
5. Understand up front whether there's any regulatory or interpretability requirements that need to be considered.
   - ↳ These requirements dictate what's an acceptable label & features, & what types of modeling algorithm should be used.

- After documenting the constraints move to DESIGNING PHASE
- COMMON DESIGN DECISIONS:

Decision	Performance Failure	Execution Bottleneck	Societal Failure
Translation of problem goal to label	Med Risk		High Risk
Choice of Tech Stack	Med Risk	High Risk	
Sample size used	High Risk	High Risk	High Risk
Features created	High Risk	High Risk	High Risk
Model candidate set tested	High Risk	High Risk	
Performance metrics used	Med Risk		High Risk

↳ Involve the model selection & evaluation stages

- We often rely on observable proxies to serve as labels.
  - ↳ Proxy ISN'T CORRELATED with a real problem ⟹ lead to SOCIETAL & PERFORMANCE FAILURE
  - ↳ EX) Using data scraped from a resume to predict if a candidate would be a good employee
    - The proxy label would be whether previously hired employee reached a certain tenure.
    - This could bias the model to only prefer candidates with certain genders or ethnic backgrounds because of underlying biases in the training data.

- Refers to the choice of computational resources and tools used to prepare the data, train the model, and deploy it.
  - ↳ No sufficient computational resource ⟹ less likely to efficiently search through an appropriate set of model candidates ⟹ less performant model from a prediction perspective

MODEL DEVELOPER BEST PRACTICES

Model developer best practices to reduce ML based risk:

1. PRACTICE AGILE MODEL DEVELOPMENT
   - ↳ Process of increasing your model complexity in separate efforts as illustrated by the curve:
   - ↳ The first model [Model v1] should be intentionally simple ∴ few simple features & model algorithm like logistic regression that doesn't consider much tuning.
   - ↳ Overtime you'll learn more about your problem & the feedback should help you build more complex versions that are stable & lower in various risks.

2. ALWAYS APPLY UNIT TESTS WHERE APPLICABLE
   - ↳ UNIT TESTING is a software development process in which the smallest testable parts of an application called units are individually and independently tested for proper operation.
   - ↳ Particularly applicable for data preprocessing stages
   - ↳ Unit tests on DATA PIPELINES:
     - → Test that data types match prescribed data schema
     - → Ensure all values of a feature are within acceptable range
     - → Assert data transformations yield the right row or column count.

3. WRITE REPRODUCIBLE CODE & PROCESSES
   - ↳ It's pretty common to find errors in your work that require you to run through a set of steps again. This might be caused, for instance, finding upstream issues in your data.
   - ↳ Having reproducible data and model pipelines will make any rework necessarily seamless & fast.
   - ↳ Reproducibility is a kind of insurance against future issues.
     - If at any point you encounter performance or regulatory problems, being able to reproduce a model can help you debug exactly what's causing these problems.
   - ↳ Writing reproducible pipelines takes upfront planning & work

4. CREATE GOOD DOCUMENTATION
   - ↳ Documentation begins in the planning phase where ideally you'll create a strong project brief that outlines the motivation from building a model and the methods & data you'll use to build it.
   - ↳ Having easy to read and documented code will help with peer review and collaboration.
   - ↳ Someone should be reviewing your model's strengths & limitations is a great way to ensure that you might ultimately deploy it doing what it's intended to do.

EXECUTION BOTTLENECK: a data or modeling process that's not terminating in a practical amount of time.

- Deciding what's an acceptable run time limit is up to you & your general problem needs.
- CAUSES OF EXECUTION BOTTLENECK:
  - → Too much data relative to your computational resources.
    - ↳ Yields 2 solutions:
      - ① Reduce your data size
        - ↳ You can reduce in both data preprocessing steps by either downsampling or using batch processing
        - ↳ DOWNSAMPLING: process of taking random subsets of your data and running the full lifecycle on the sample data.
        - ↳ In the process of developing and testing code we typically use a sample of data to help avoid runtime bottlenecks. When were satisfied with the code, run it on the full data set.
        - ↳ PERFORMANCE TRADEOFF: smaller data sets may reduce your performance potential (predictive potential)
      - ② Batch processing
        - ↳ Defined as reading partitions of the data into memory & perform data operations on each partition, one at a time.
        - ↳ NOT ALL ML algorithms accommodate batch processing during the TRAINING step. This is normally available in LOGISTIC REGRESSION or NEURAL NETWORKS where we use gradient optimization techniques to fit model weights that minimize the loss functions.
      - ③ Keep all the data you have BUT ADD more COMPUTING POWER
        - ↳ More of an option at work than at home ∵ computing power cost lots of money

④ Parallelize your processing as much as possible
↳ PARALLEL PROCESSING:

First assume we have a set of independent computer processors. These can be separate servers or processors within a given computer

The data gets either REPLICATED or SPLIT to each processor.

Each processor performs some compute logic on the associated data.

After each processor finished its operation, some desired output is sent to a central server for aggregation & further processing.

↳ There's a framework called MapReduce that's often used to describe this whole process.
↳ The MAP step is where data & code are sent to be processed in parallel.
↳ The REDUCE step is where the aggregation of the parallelized results take place.
↳ Many data and model processing algorithms are naturally parallelizable.
↳ Most modern database engines will use a MapReduce framework to run data processing pipelines. All you have to do is run basic SQL commands.
↳ Parallel processing is appropriate where computing on one data partition doesn't depend on the results of computations on other data partitions

Ex) Running 3CV in a parallel computing environment

Each processor is training & validating a model on the appropriate folds. The results will then be passed to the aggregator for model selection.

BIAS, VARIANCE, AND DATA SIZE

When examining the different ML failures the common cause is the SIZE OF THE DATA.

- Too much data can cause EXECUTION BOTTLENECKS & DOWNSIZING is an effective strategy to mitigate that.
- BUT at what point did we downsample too much? We can answer through BIAS-VARIANCE TRADEOFF

BIAS VARIANCE TRADEOFF is formalized way to understand the competing concepts of model over and underfitting, which are driven by a model's complexity and the amount of data we have to train.

After we train a model and evaluate its predictions, we're likely to observe some level of error in the predictions.

- No matter what loss function we use the model error, the error typically comes from 2 sources:

SOURCE 1: Model estimation bias
↳ MODEL ESTIMATION BIAS: whether or not the true relationship between the features and the label
↳ More data ⟹ estimation bias doesn't improve

SOURCE 2: Model estimation variance
↳ MODEL ESTIMATION VARIANCE: how much a model's output change if we were to train it on different random samples of our data.
↳ More data ⟹ estimation variance ↓
↳ More complex algorithm ⟹ estimation variance ↑

- More data enables us to use more complex models (i.e. logistic regression to DT) and ∴ can reduce bias.

∴ More data reduces the model estimation variance while indirectly enables us to reduce model estimation bias.

∴ Reducing both estimation bias & variance yields better generalization performance overall.

CLASS IMBALANCE

CAUSE of PERFORMANCE FAILURES that's caused by data size but is characterized as not having enough of the right data: CLASS IMBALANCE PROBLEM.

CLASS IMBALANCE PROBLEM: situation where one of our classes is much more rare in the data. This usually involves binary cases, & the class that's more rare is usually the positive class, or oftentimes what we label as one.

↳ Created by natural processes & doesn't indicate that there's an error in data collection
↳ A problem ∵ most supervised learning algorithms needs sufficient representation of both classes to learn well. Additionally when one class is overrepresented, the learning algorithm will implicitly weight the majority class more.
↳ When mitigating this problem we want to resample data so that we can retain as much of the minority class as possible while reducing the imbalance.
↳ Approach this through both: DOWNSAMPLING & UPSAMPLING.

① DOWNSAMPLING
→ Better choice when starting with very large data
→ STRATIFIED SAMPLING & STATISTICS: sampling rates are conditional on a given attribute of the data.

② UPSAMPLING
→ Better strategy when we have limited data to begin with & can't afford to discard any.

↳ To SOLVE class imbalance:
- Design decision that you can empirically test

RULE OF THUMB:
① If base rate is > 5% you don't have to make adjustments
② If base rate is < 5% you should consider other strategies.

③ To empirically test different class balancing strategies, always use a validation set that has the original class distribution.

↳ RECALL: metrics S.A. accuracy, precision, and recall will change if base rates of your data changes which ultimately happens when you do undersampling & oversampling.

∴ you can't compare these stats on 2 data sets with different class ratios

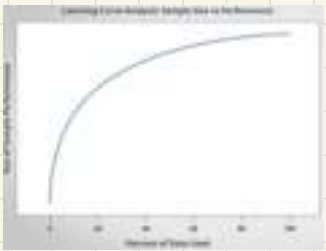
HOWEVER: When testing different class balancing strategies that's exactly what your doing. Keep your validation data in its OG form & compare different strategies against this data.

↳ Mitigating class imbalance can overall improve results.

LEARNING CURVE ANALYSIS

Method to empirically measure data size efficiency: LEARNING CURVE ANALYSIS

LEARNING CURVE ANALYSIS: Plot that shows the relationship between data size & model prediction performance.

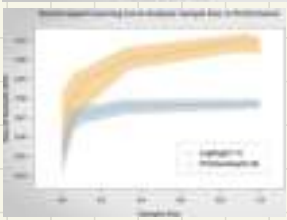


To build a curve like this:

STEP 1: Select a range of sampling rates to cover the x-axis. Usually you sample up until your max data size.

STEP 2: Run a loop where each iteration you sample from your training data, build a model, & then evaluate this model on your test data.

↳ Test data should've been created before the main loop, which you use to build the curve.



- Logistic regression is a simple linear model. It has lower variance which is why it can perform as well w/ only 10% of the data.

But no matter how much data we give it, it can't reach the performance levels of the DT.
∵ LR inherent model estimation bias.

If your curve looks more like the DT instead of LR here, you may want to explore investing in more data.

↳ Learning curve analysis will help you identify the right tradeoff between execution speed & predictive performance.

IRRELEVANT FEATURES

- Model feature is heavily dependent on your choice of features
- When you have irrelevant features, its best to remove them using standard feature selection techniques.
- Irrelevant features can cause 2 CORE PROBLEMS:
  1. Increase the risk of overfitting ⟹ prediction performance failure
  2. Create engineering & maintenance costs ⟹ don't incur cost that generate no benefit

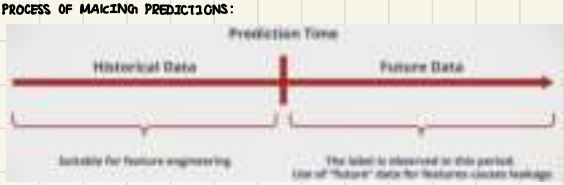
FEATURE LEAKAGE: when you use information in the model training process that would not be available at prediction time.

↳ A cause of model performance failures.

↳ It can lead model developers into false sense of accomplishment.

↳ To understand this problem, we need to go back to the FEATURE ENGINEERING PHASE.

↳ PROCESS OF MAKING PREDICTIONS:



→ When preparing our data for MODELING, the label should be built from events that happen in the future time period.

→ All features should be built using data from the time period marked here as future data, you would be committing LEAKAGE ERROR.

↳ EX) Assume you're working at an e-commerce company & you want to build a model that predicts whether a visitor to your website is going to make a purchase. The model will be run in real time on the site & you might use the predictions to customize the offers that you present to the web visitor with each click.

Your features will be built from different portions of the website that the customer has already clicked on.

Any web click that happens BEFORE a purchase occurs is eligible to use as a feature.

Assume that in your data you accidentally left purchase confirmation pages & use that as a feature. This would be considered leakage ∵ the event happened AFTER we observed the labels, which is the event that someone made a purchase.

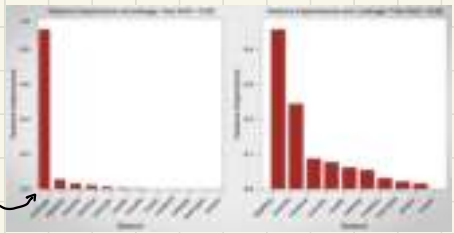
→ If leakage was included in the model this would lead to exceptional predictive performance in the offline analysis.

REMEMBER when testing for overfitting we usually want to see how strong training performance but BAD OUT-OF-SAMPLE performance.

→ Leakage impacts both training & out of sample data set.

→ DETECTING LEAKAGE:

1. High out of sample performance (too good to be true)
   ↳ When AUC > ~90% then there's something suspicious going on.
2. Feature importance:



A feature with much relative feature importance

3. Thoroughly look at your code to make sure you haven't inadvertently made mistakes handling the timestamps of the data, particularly for feature engineering, labeling as well as splitting your training & test data sets.
4. Perform FEATURE IMPORTANCE ANALYSIS to identify BOTH irrelevant & leak features as well as help us create a narrative about how our model is actually working.

CONCEPT DRIFT: changes in the statistical properties of data overtime.

→ STATISTICAL PROPERTIES: can mean any one of the following: mean value of the feature or label can be changing over time
variance of the feature or label can be changing over time
expected value of the label, conditional on the features, can be changing over time.

→ Overtime the both features' mean and variance changes overtime.

In general when our data is being generated by regular, everyday processes, we should always expect that the features will change overtime and the relationships between the features will change overtime & the relationship between the features & labels should change overtime.

→ Businesses in general changes will happen in the company, growth, competition, marketing, etc. can change the distribution of the data you can collect.
The KEY is to build systems that are resilient to such changes. This is acheived through:

1. Regularly monitoring your model
   → Build and implement systems that track 2 things:
   A. Mean and variance of your input features
   B. Actual predictive performance of your model over time. [TOP CONCERN]
      ↳ If this starts to drop you need to make plans to remediate the problem.
      ↳ Best remediation is to retrain your models on more recent data.
      ↳ When we have a lot of historical data we need to balance a certain tradeoff.
      Using more historical data gives us more data, which we know helps to reduce model estimation variance, which reduces overfitting.
      HOWEVER data that's further away from the current timestamp may be less relevant. This creates a certain type of distribution bias, which is the same bias we encounter with concept drift
      We can set up an experiment that'll guide us on how much historical data we should actually use.
      We first set aside the most recent periods for validation and test data.
      Then each variant of our experiment, we use a different length of historical data, starting w/ the most recent & then going backwards
2. Regularly retraining your model
   → Retraining a model takes time & development cost. The cost of retraining needs to be balanced against the cost of degregation.
   → If you set up the training procedures to be reproducible then the cost of retraining should be much lower than the OG development cost, which gives you better performance overtime.

ALGORITHMIC FAIRNESS

SOCIETAL FAILURES

→ General focus of ethical AI

→ EXAMPLE #1: ProPublica investigated the use of a commercial tool called COMPAS which predicts whether a criminal defendant might fully commit crimes based on their criminal history & other factors. The predictions are used by judges to make sentencing & bailing decisions. ProPublica's investigation found that the model errors were different for different races. The study found that black defendents had FP rate 2x higher than white defendent. FP can harm defendents.

COMPAS tool company Northpointe responded with a counter analysis where they claimed that their tool was fair.

This debate resulted in several research on different ways to measure algorithmic fairness & often these metrics are incompatible with each other. Meaning you can acheive fairness with 1 metric.

ProPublica used 1 fairness metric and Northpointe used different ones which led to opposing conclusions that were correct by their own standards of fairness [ALLOCATED & REPRESENTATIONAL HARM]

→ EXAMPLE #2: Accuracy of facial recognition systems when comparing people with different skin tones. Studies show that people with darker skin, particulary dark skinned women. This was a major cause for concern ∵ commercial facial recognition systems are widely used for a variety of commercial & government tasks. [ALLOCATED & REPRESENTATIONAL HARM]

↳ **ALLOCATIVE HARM**: Discriminatory system that withholds certain opportunities, freedoms, or resources from specific groups.

REPRESENTATIONAL HARM: system reinforces negative stereotypes along the lines of identity & protected class.

↳ Fairness is centered around whether the system is able to cause specific harms, both by directly disadvantaging people or providing advantages to some people over another.

Several fairness discussion centers on model performance disparities between groups usually defined by identification along some predicted class.

↳ A lot of fairness starts with your awareness of the potential harms that the systems can cause as well as your ability to develop models that generalizes well.

METRICS AND TYPES OF FAIRNESS

Improperly implemented ML models can cause societal harm that are detrimental to underprivileged and underrepresented groups.
MLE job is to be aware of such harms & mitigate them if not eliminate them.

2 COMMON HARMS CAUSED BY UNFAIRNESS IN ML:

1. **ALLOCATIVE HARM**
   - Concerned with the idea of allocating resources.
   - ML models have the potential to serve all groups of people equally regardless of their gender, race, & religion.
   - When a model prefers one group over another it becomes harmful to society.

   Ex) Approving a loan higher rate to people of certain race.
   Ex) Making positive hiring recommendations only to people of certain gender.

2. **REPRESENTATIONAL HARM**
   - Concerned with the idea of how reality is presented & hence shaped by a ML model.

   Ex) Recommendation engine showing harmful stereotypes of certain ethnic group
   Ex) Represent certain groups as less likely to become CEOs.

ADDRESSING ALGORITHMIC FAIRNESS

Mitigation techniques for fairness concerns we might discover in our ML models.

Mitigation techniques to integrate into standard model development process:

STEP 1: PROBLEM FORMULATION
↳ Define a fairness goal (fairness goal can be defined in numerous ways)
↳ Different fairness definitions often compete with each other.
↳ Determining what's best is something you'll discuss with your peers. You'll want to be inclusive as possible.
↳ Include/consult with more people. People of different communities have different experience & values.
↳ When CREATING THE LABEL, you'll need to be fair.
- Creating labels require subjective judgement
- Labels could encode biased decisions or corporate cultural norms that the company regularly practices.
- When encoding a bias into a model, we perpetuate it at scale, which is exactly the type of thing we're looking to avoid.

STEP 2: DATA PREPARATION
↳ SOURCE of algorithmic bias: individual features are HIGHLY CORRELATED with a protected class attribute

Ex) zip codes are a good proxy for an individual's race or ethnicity. ∴ using zip code in a model is as effective as using race.

↳ Protected class attributes and proxies (S.A. zip codes) are explicitly forbidden from use in model.
↳ EFFECTIVELY USING PROTECTED CLASS DATA IF YOU ACTUALLY HAVE IT.

The most important use of such data is to run fairness evaluations of your model.

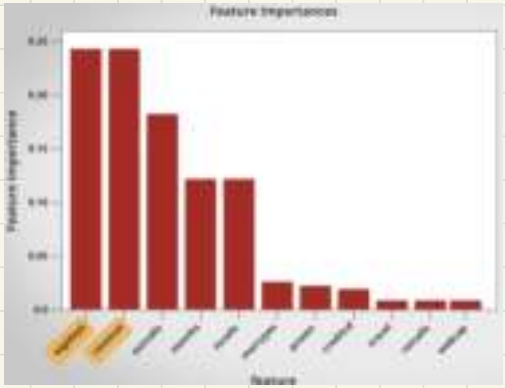
↳ FAIRNESS OBJECTIVE: evaluate your model against the fairness metrics that are appropriate given your goals for fairness and for the different class attributes that are available for you.
↳ Many companies make a point to NOT collect these information ∵ it's explicitly forbidden or the privacy and legal risk is too large.
↳ These attributes are often not collected in order to preserve people's privacy and to protect people from explicitly discriminatory practices. But algorithmic bias happens unintentionally, and the same efforts taken to prevent explicit discrimination makes it challenging to detect implicit or accidental discrimination.

ALGORITHMIC ACCOUNTABILITY: model owners and developers are accountable for the decisions that their ML systems make.

↳ Covers all impacts of decision making from PREDICTIVE PERFORMANCE, SYSTEM RELIABILITY, TO ALGORITHMIC DISCRIMINATION.
↳ To achieve algorithmic accountability:

1. CREATE TRANSPARENCY AROUND HOW OUR MODELS ARE PERFORMING
   - Understanding how a model maps inputs to predictions can help with several concerns.
   - HIGH LEVEL: provides insights into which features are worth investing in & which should be left out
     ↳ These insights shape how we understand the core problem we're trying to model from CAUSE & EFFECT prospective.
   - MICRO LEVEL: algorithmic transparency helps us understand why specific decisions or predictions were made for specific examples.

↳ Model transparency involves FEATURE ANALYSIS.
- Start by looking at feature importance derived from tree-based models



- This chart shows the 2 most important features: EQP DAYS and REVENUE
  - EQP DAYS = # of days that a customer has had their phone equipment
  - REVENUE = average monthly amount the customer pays

→ Feature importance offers limited insight into how these features actually impact the label.

→ To gain more insights: **PARTIAL DEPENDENCY PLOT**

→ Once we know that a certain feature is important we like to know the relationship between that feature and the outcome.

Simple way: x (↑ FEATURE) vs. y plot } ISSUE WITH SIMPLE PLOT: Brings the idea of: CONFOUNDING

→ Partial dependency plot can solve the confounding problem.

It performs the technique: **COUNTERFACTUAL ANALYSIS**

FACTURAL ANALYSIS: measuring x and y from the observed data

COUNTERFACTUAL ANALYSIS: What would the label look like if everyone had a certain value of this particular feature instead of the one they had to observe.

↳ Creating partial dependancy plot:

1. Define a range over the feature instead of the points in that range.
2. For each value in that range we give all examples in our data the same value, make predictions on our model, and then take the average of these predictions.

→ REMEMBER: To compare the partial dependency plot with simple x vs. y plot is the partial dependency plot tells us how the model we are using relates features to an actual outcome or a prediction. This is what enables us to get some high-level view of how the model is operating.

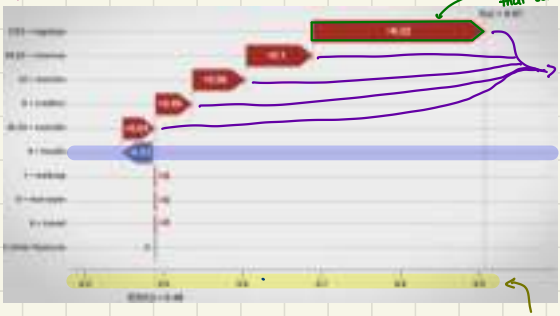
If we use this to understand individual predictions we can look up the feature values for a given example and see how sensitive the model is around that range of the feature.

When there's a lot of data = relationship between x and y is MORE RELIABLE
When there's less data = be skeptical that the actual relationship that has been modeled would generalize well.

→ If a particular example had feature example had feature values that were in the low density regions, we would expect the model to be more wrong on average.

→ Creating **INSTANCE LEVEL MODEL** methods:

1. LIME
   - Aims to measure how much individual features influence the output of the model @ granularity of a single example.
2. SHAP
   - Aims to measure how much individual features influence the output of the model @ granularity of a single example.
   - EX) Waterfall plot with the SHAP python package:



This example has a high churn risk that can be attributed to high values in the revenue and EQP days feature.

- Reads from BOTTOM to TOP

Notice how the direction and influence of individual features is consistent with our feature importance

The bars show how each feature for this example is contributing to a change in the prediction where that change is relative to the average churn.

Average churn rate for all examples in our data.

- EX)

- The churn risk is MUCH LOWER than the PREVIOUS
- The EQP days is still influential BUT the value brings the churn risk DOWN instead of up.
- There's some influence from the features outcalls which brings the overall risk ↓

OVERALL... 2 global methods: ① Examine feature importance to get relative sense of what features mattered

② Partial plot dependency plots: provides granular detail on how a feature relates to the particular outcome.

③ Instance level explanation: SHAP; which provides explanations for individual examples showing how the values of the features at that example changed the prediction of that example.

These tools will help you provide insights while creating a sense of accountability so that you can be more assured that you're protecting your customers & company from any unintentional biases & discrimination.

ASK THE EXPERT:

Q1: WHAT DOES IT MEAN TO "DEPLOY YOUR ML MODEL"

- ML MODEL DEPLOYMENT is the process by which a ML algorithm is converted into the web service & then into an application
- We refer this conversion process as the OPERATIONALIZATION or DEPLOYMENT OF THE ML ALGORITHM.
- OPERATIONALIZING a ML model really means to transform it into something consumable. Transform the model you build into AI application.
- With this deployment process other people can call your model & produce results out of it.
- The moment when ML becomes AI.

Q2: DIFFERENT TOOLS, PLATFORMS, AND ARCHITECTURES USED IN THE INDUSTRY

- Python for building, training, testing, and deployment of ML models.
- Cloud computing and their typologies of services which provides an environment to run your code & have access to storage to store data, to end data pipeline (meaning you get new & refreshed data at specific intervals).
- Data visualization for the opportunity to understand your data & communicate your results
  - ↳ We can use POWER BI and TABLEAU, PYTHON'S MATPLOTLIB

Q3: HOW TO DEPLOY A MODEL?

- 2 main functions that you need to use: init() and run(input_data) which are the CORE part of model deployment.
  - ↳ init function:
    - Focuses on preparing your data because as you know after the data preparation, the model training, testing, validation, you select a model & then you deploy it, but the data preparation part has to be there because any different typologies of data you've going to use, any new data that you're going to use, then it needs to be prepared in the same way.
    - All about preparing your data & ensuring that the input data that you're using is going through the same process.
    - This data will then be used to feed your ML model.
  - ↳ run function:
    - Makes sure that your model knows how to read the data, how to be run in order to produce your results.
- You can use different tools & technologies that helps you pack these 2 functions into a file: PICKLE FILE
  - ↳ Capture information that was referred to earlier: how your data needs to be prepared
    how your data is going to be read by the model so that the model can produce the predicted results that you want.

Q4: WHAT ARE THE STEPS INVOLVED IN THE DEPLOYMENT PROCESS?

- STEP 1: Registering your model
  - ↳ Ensures there's a sort of a logical container for one or more files that than you are going to use for running your model.
  - ↳ For instance you can register multiple files and register them as a single model in your ML workspace. This is an environment for collaboration & experimentation.
- STEP 2: Preparing your models to be deployed
  - ↳ Specify different assets, the usage, & the computer target that you're going to use.
- STEP 3: Deployment
  - ↳ Deployment to computer target

Q5: HOW DO YOU DETECT ERRORS IN DEPLOYMENT?

- The init function ∵ this function loads the model into the global object & is the function that runs only once. The error you receive will indicate that your data is not processed in the right way.
- The run function at the input data moment.
  - ↳ This function uses the model to predict a value ∵ we are in a ML scenario, based on the input data.
  - ↳ We also use a process called the SERIALIZATION & DESERIALIZATION this means that the inputs & outputs to run typically are JSON for serialization-deserialization.

Q6: WHAT ARE SOME COMMON ROLES IN THIS FIELD?

- DATA SCIENTIST:
  - ↳ Focuses on the modeling part of the end-to-end ML life cycles
  - ↳ ML life cycle:

Data scientists needs to know all 4

STEP 1: Business understanding
STEP 2: Data exploration, Data acquisition
STEP 3: Modeling
STEP 4: Deployment

DATA ENGINEER
Making sure that the different data sources that you're going to use is ready to be accessed & is also part of the data architecture, a data pipeline that's stable so there's no interruptions in the data pipeline.

DATA SCIENTIST
Focuses on feature engineering, picking the right algorithm, then making sure that all the hyperparameter tunings are done — all the processes around optimizing the algorithm are completed so they can pick the best model that's performing the best for their own scenario with their own data.

MACHINE LEARNING ENGINEER
Responsible for deployment & the end-to-end solution. All of the deployment, transforming your ML model, ML algorithm into an AI application is done by ML engineers.

- AI ETHICS SPECIALISTS / AI FAIRNESS SPECIALIST:
  - ↳ Has extensive experience on how to use different tools in order to assess AI systems' fairness & how to mitigate unfairness issues.
  - ↳ To assess AI fairness:
    - ① Fairlearn packages

Q7: HOW DOES ML TEAM CONTRIBUTE TO BUILDING BETTER & UNBIASED MODELS & FAIR ML SYSTEMS?

- Diverse team is important ∵ those who build AI & ML & ML algorithms & the consequences these AI applications that are going to have a direct impact on other people's lives.
- Diversity brings different perspectives & a variety way of solving problems & different types of algorithms used.
- Diversity helps your team build better models.

Q8: WHAT ARE AI ETHICS?

- AI systems are deployed to impact several decisions that are being made for us as humans.
- AI raises concerns about its potential of being unfair and being trained on historical data from society. It has the potential to not be transparent & for you to completely lose track of how it has made its predictions.
- There's potential in losing the privacy of individuals while making decisions for them.
- RESPONSIBLE AI: critical consideration around AI transparency, reliability, fairness, privacy, & accountability & provides companies with the best practices, set of tools, guidelines, and framework to follow to ensure AI is not harming humans.
  - ↳ Can guard against the use of historically biased data to train models, ensure that automated decisions are justified & explainable & ultimately help maintain user trust & individual privacy.
  - ↳ To help provide a clear set of rules & best practices to companies & allows them to innovate & realize the transformative potential of AI while staying 100% accountable for the impact it has on humans.