Week 3, 01-21-22

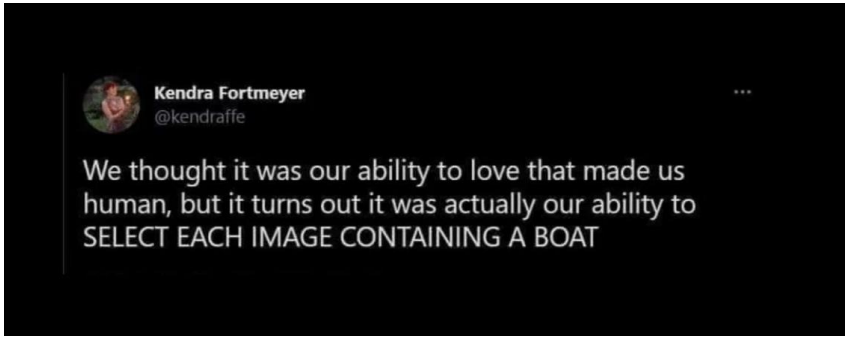
CS148 Discussion Section 1C

Checking in...

- So like we're maybe, possibly, going to be in-person again...

- Project 1 in the can!
- HW1 due January 26
- Project 2 Released, will be released on January 26 as well

- Student Directory reminder!



Pajama Party!

- Let's do this thing!
- Poll to vote for best Pajamas here:
  - https://forms.gle/SWHaCXS3AFqwLzMN9

Coding Sample

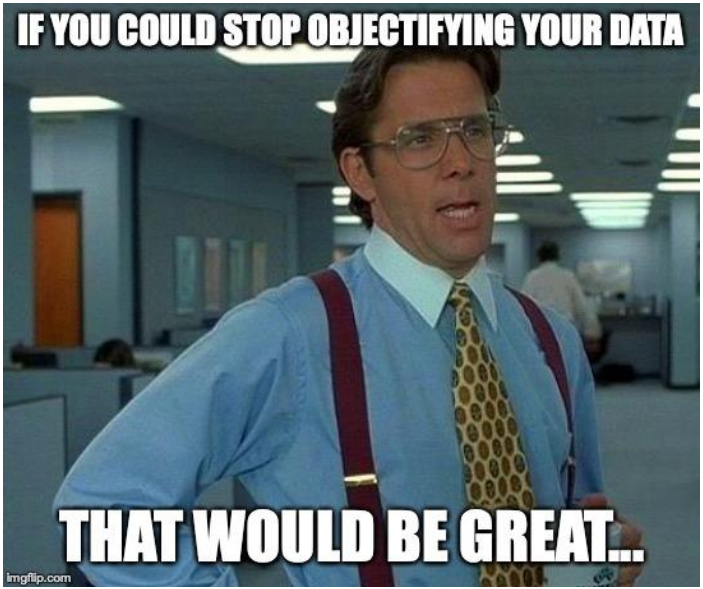
Sample Code Runthrough!

- We're doing avocados!
  - Let's do two examples:
    - Predictive Regression to predict the average Price of an avocado
    - Classification exercise to predict Organic vs. Conventional
      - Preparation for Project 2!
- Jupyter Notebook and Data Available here
  - Save them in same directory or update file path in the notebook

Objects are Evil

```
In [3]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
Unnamed: 0      18249 non-null int64
Date            18249 non-null object
AveragePrice    18249 non-null float64
Total Volume    18249 non-null float64
4046            18249 non-null float64
4225            18249 non-null float64
4770            18249 non-null float64
Total Bags      18249 non-null float64
Small Bags      18249 non-null float64
Large Bags      18249 non-null float64
XLarge Bags     18249 non-null float64
type            18249 non-null object
year            18249 non-null int64
region          18249 non-null object
dtypes: float64(9), int64(2), object(3)
memory usage: 1.9+ MB
```



- Remember! Only INT and FLOAT types can be input into a model
- If you see an object, plan on either dropping or converting.

A Wrinkle in Time

```
In [5]: data['Date'] = pd.to_datetime(data['Date'])
        data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
Unnamed: 0      18249 non-null int64
Date            18249 non-null datetime64[ns]
AveragePrice    18249 non-null float64
Total Volume    18249 non-null float64
4046            18249 non-null float64
4225            18249 non-null float64
4770            18249 non-null float64
Total Bags      18249 non-null float64
Small Bags      18249 non-null float64
Large Bags      18249 non-null float64
XLarge Bags     18249 non-null float64
type            18249 non-null object
year            18249 non-null int64
region          18249 non-null object
dtypes: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 1.9+ MB
```

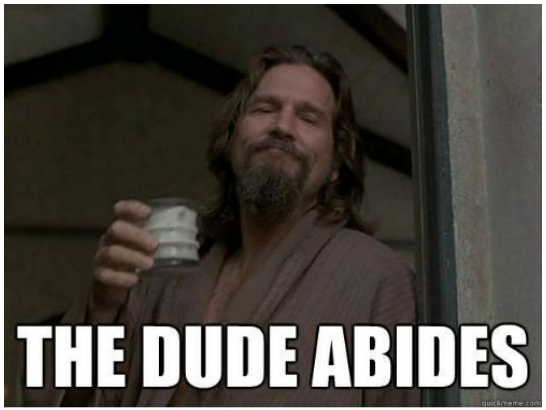
- Really cool date/time function and great for augmenting features!
  - E.g., month/season, total volume sold during the previous intervals, avg price over the last month
- Not relevant for our projects so we're just going to point it out as we drive on by!

The Dude's Certified Best Formula for Processing Your Data

1. What matters? Are you Pipelining your data or not!
    a. Why????
        i. The pipeline is effectively part of your model that you want to test and the training values you fit on will be applied to all future data
2. If not pipelining:
    a. Copy your labels to a new array and then drop them from your dataframe
        i. Why????
    b. Drop all the features you aren't going to use
    c. Fix the rest:
        i. Check all features are floats or ints (convert as necessary)
        ii. Imput missing data
        iii. For Categorical
            1. LabelEncode, Ordinal encode, or OneHotEncode
        iv. For Numeric
            1. Scale if wanted, or pass through as is!
        v. Engineer or augment additional Features
    d. Split your data
    e. Train
    f. Test
    g. Score it!

The Dude's Certified Best Formula for Processing Your Data (cont.)

1. If pipelining:
    a. Copy your labels to a new array and then drop them from your dataframe
    b. Split your data into training and testing cohorts
    c. Arrange a Pipeline
        i. Determine features to include and bucket them
        ii. Check all features are floats or ints (convert as necessary)
        iii. Imput missing data
        iv. For Categorical
            1. LabelEncode, Ordinal encode, or OneHotEncode
        v. For Numeric
            1. Scale if wanted, or pass through as is!
        vi. Engineer or augment additional Features
    d. Pipeline the Training data (fit_transform())
    e. Train the model
    f. Pipeline the Test data (transform())
        i. How do we handle categorical mismatches?
    g. Predict() on the test data
    h. Score it!
2. So why not always pipeline? Don't need it, and can get messy when cross-validating



Remember: Never Leave your Labels in your Features!

- Actual footage of your model classifying with the labels still in the data

Quick Note: Alternative approach(es) to splitting data

- Introducing: StratifiedShuffleSplit
  - Key idea: Simply splitting the dataset with train_test_split leaves the train and test set vulnerable statistically unrepresentative sampling
  - Stratifying attempts to correct this by ensuring that the sample is 'random', but still maintains proper ratios within test and train splits based on a predetermined category!


```
from sklearn.model_selection import StratifiedShuffleSplit
# let's first start by creating our train and test sets
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    train_set = housing.loc[train_index]
    test_set = housing.loc[test_index]
```

  - From our AirBNB data, can you think of an example?

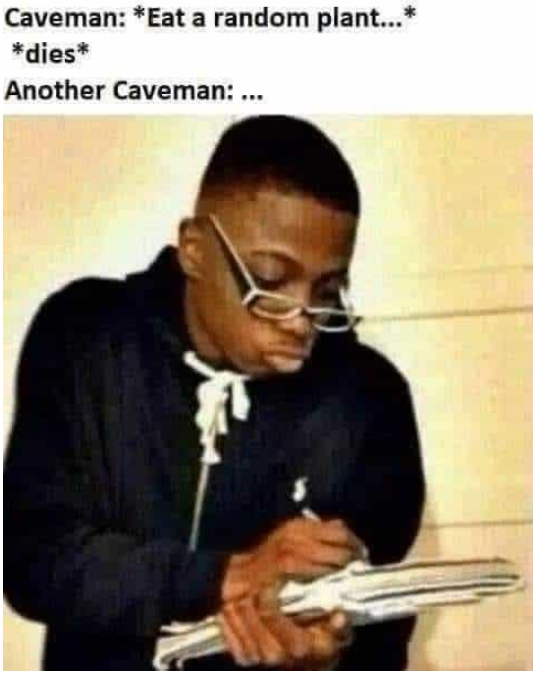
HW1

Data Engineering - 'Fixing Your Data'

- What issues are intrinsic to your data collection
  - We have a natural inclination to view data, being numeric in nature, as being accurate. But it is just as messy and error-prone as anything else
  - Q1 is designed to explore this
- Translating Questions into 'data' and interpreting relationships
  - Q5 focus on this!
- Inserting data into Model
  - Am I modeling the relationship between my features and labels correctly?
    - A simple linear regression works when I have an ordinally structured variable (numeric or categorical) that moves in consistent relation to the label
  - Does data work this way?
    - Often not! Relationships can be messy

$$y = mx + b$$

O.G. Datascience...

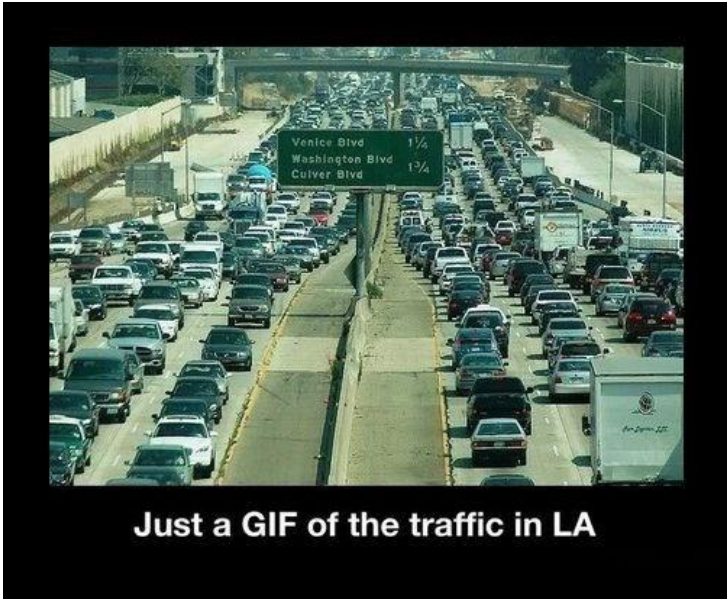


Example: LA Traffic

- Classic ML Problem: Predict drive-time!
- Let's consider 3 example features:
  - Location (in longitude and Latitude)
  - Time (Continuous numeric feature)
  - Direction Traveling (categorical Compass)
- Problem: All seem useful, however the relationship between these features and the label (time in traffic) seem messy and non-linear
  - **Location**: Traffic trends will almost never trend uniformly over a directional axis but will instead cluster around key commercial and residential clusters
  - **Time**: Will not be uniformly trending, with certain intervals (rushhour) seeing peaks and ebbs
  - **Direction**: Traffic is typically highly related to directionality but also time sensitive. Traffic into and out of urban centers will see dramatic differences during morning and evening rush hour

LA Traffic Cont.

- Some strategies we can take:
  - ‘Blocking’ continuous variables into discrete categories
    - **Location**: Oftentimes it is more useful to look at location in aggregate as a neighborhood or region. Augment a ‘neighborhood’ feature to block regions together for discrete analysis
      - Same thing with time! ‘Morning’, ‘Afternoon’, and ‘Evening’ categories may prove more useful
    - **Direction X Time:** Here we know these variables are intrinsically interconnected, so let’s capture that dynamic by modeling the crossterm of these two instances:
      - E.g., (Heading East x Morning), (Heading North x Evening)
        - Let’s drill down on this further…



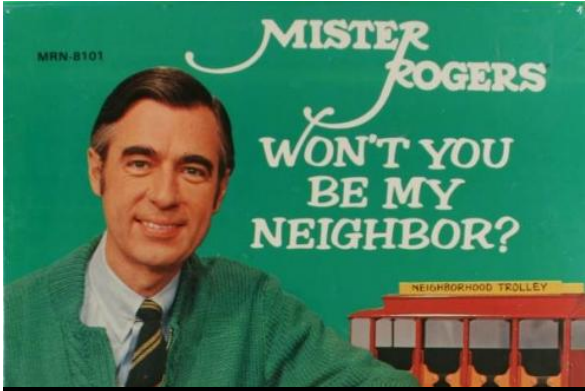
- Another cool thing we can do? **Period Regressions**! I.e., instead of one model for the entire day, segment the data and develop multiple models!

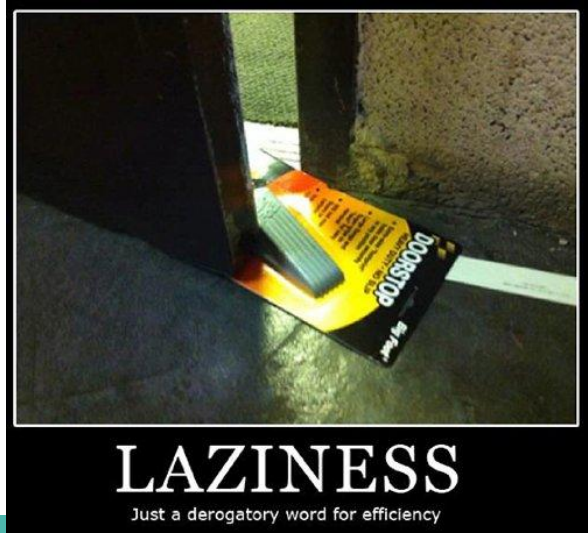
What is One Hot Encoding Doing Really?

- We've talked about it in terms of what it avoids (i.e., not introducing an artificial hierarchy to our dataset where there is none), but haven't really expounded on what makes it so cool!
- When you have a category of functionally independent values, each of which is likely to have an impact on our label when present, OHE allows us to capture and model that distinct impact
  - E.g., Fashion Brand (feature) on Clothing price
    - Each label clearly has a multiplier effect on product price, but it's not a direct linear one
      - By OHE, you allow your model to train the specific coefficient associated with EACH fashion label
    - Our crossterm of direction and time allow us to model the unique impact of each of those sets of circumstances
      - Overall can be very useful! But can come at a cost of dimensionality blowout and overfitting

K-Nearest Neighbor

- KNN is a **non-parametric**, **lazy** learning algorithm. Its purpose is to use a dataset (not a model!) in which the data points are separated into several classes to predict the classification of a new sample point.
  - **Non-parametric** meaning that it does not make any assumptions on the underlying data distribution.
  - **Lazy** meaning it does not use the training data points to do any generalization. In other words, there is no explicit training phase, and our algorithm does not 'learn a model' about the underlying data.



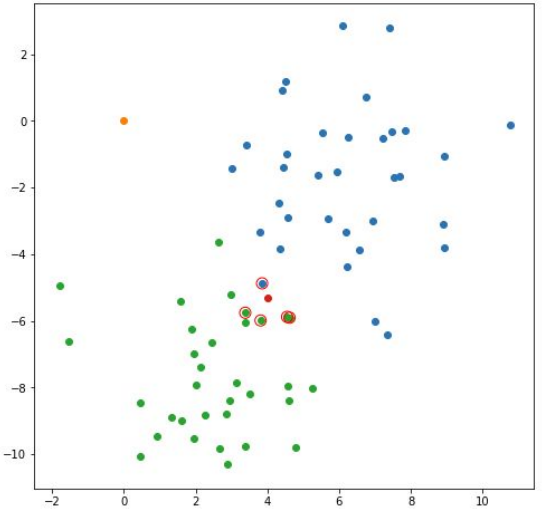


So how does a KNN Model Work?

1. We load our ‘training’ data into our model
2. A positive integer ‘k’ is specified, along with a new ‘test’ sample
3. We select the k entries in our database closest to the new sample
4. We find the most common classification of these entries
5. This is the classification assigned to the new sample
    a. In the case of prediction, the value is taking the average of the closest neighbors

A few other features of KNN:

- KNN stores the entire training dataset which it uses as its representation.
- KNN does not learn any model.
- KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.



A KNN Model with a K-Value of 5

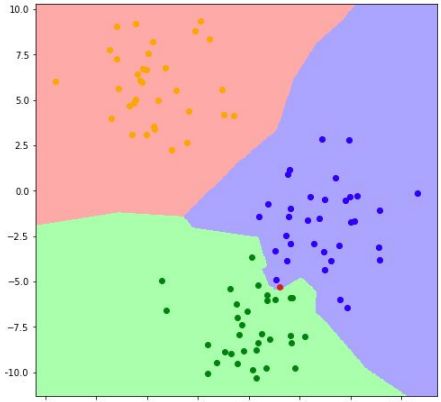
Advantages / Disadvantages to KNNs (& Clustering Generally)

- Advantages:
  - We can immediately begin classifying once we have our data! (No training needed!)
  - Generally very useful for understanding basic structures of data
  - Fairly robust method for achieving good classification
  - Can work very well on non-linear or multi-clustered data
  - Intuition: Datasets with different ‘clusters’ will do well with Clustering Models
    - E.g., types 1 and 2 diabetes, or different fraud types
- Disadvantages:
  - Have to keep the entire training set in memory to classify, so computationally expensive
  - Does not scale, or allow for easy production deployment
  - Since the algorithm does not learn a model, it doesn’t weigh the dimensions appropriately, so heavily impacted by scaling issues
- For these reasons, KNN tends to work best on smaller data-sets that do not have many features, and tend to be non-linear.

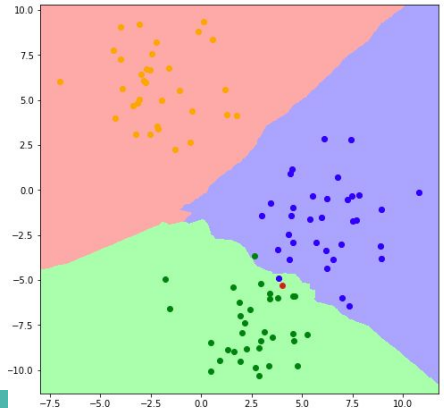
Parameter Tuning for KNNs

- K - Neighbors
  - The number of nearest neighbors that one uses
    - Higher values of k will smoothen decision boundaries
    - May be helpful for avoiding overfitting and improve generalization, but may also compromise the efficacy of the decision model
      - **Keep this point in mind...**
  - Generally (but not always!) choose odd numbers to avoid ties
- Weights
  - Weighing the neighbors can also be useful by assigning greater importance to closer datapoints
- Distance
  - Most common options are Euclidean and Manhattan

Decision Boundary with k =1

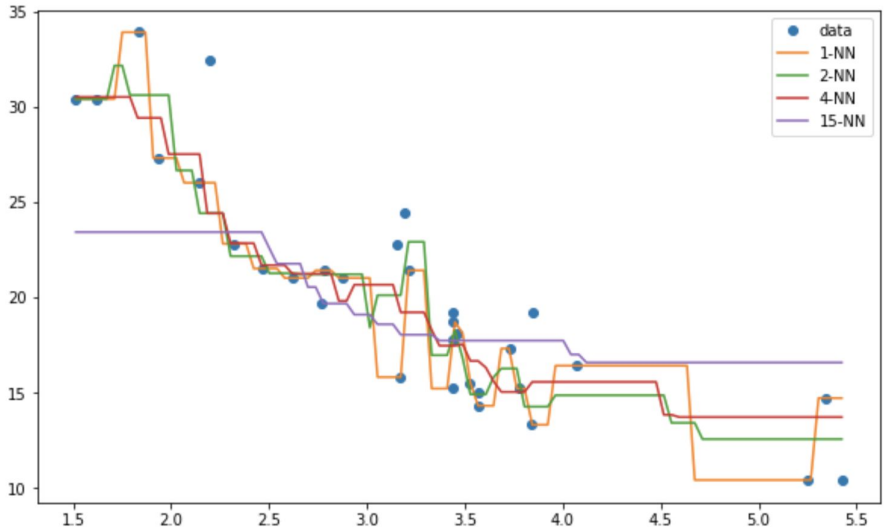


Decision Boundary with k =5



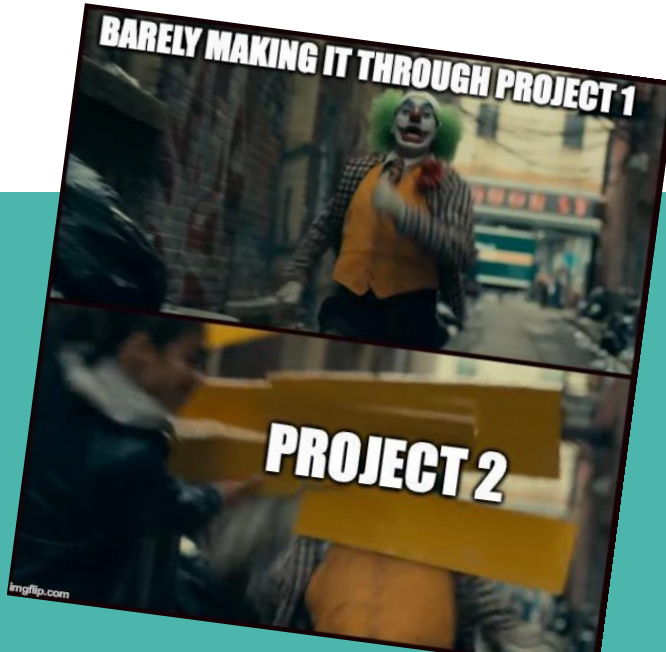
So what does a KNN Predictive Regression look like?

- Trend line between points
  - For K = 1 directly equal to point
  - For K > 1 average of nearest points
  - ‘Sphere of influence’
    - What are the KNN for this range of X values
    - Will ‘smoothen’ as more K are involved



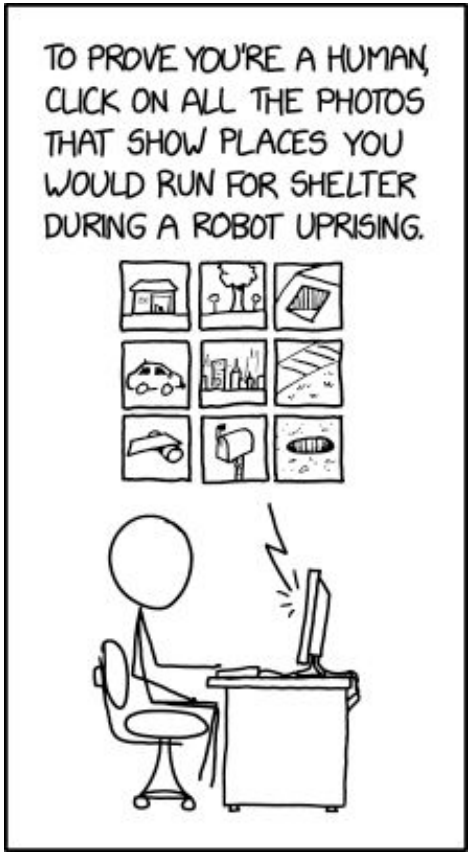
Bonus Content

Project 2



Let's Talk Classification!

- Classification is a classic ML problem
  - Image detection: Is it a cat, is it a stop sign?
  - Pattern detection: Is this a disease?
- For this project the focus is on creating the optimal model for your dataset
- Very typical of datascientist workflow
  - Once you've settled on a data processing strategy you then plug into a number of different models before finding the one that most optimally classifies your data
  - Within each model you can 'tune' your parameters to further improve performance
    - Typically you then end up going with a neural net anyways...

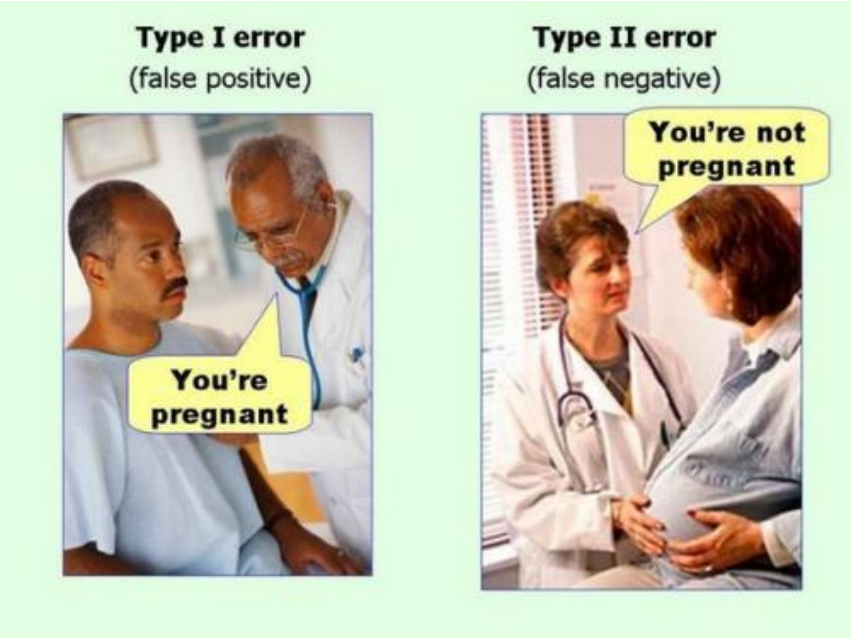


Project 2 in a Nutshell

1. First we want to demonstrate just how powerful processing your dataset is, by plugging a ‘raw’ data and a ‘processed’ dataset into the same learning model
2. Then we plug the processed data into a number of models
   a. We tweak parameters and learning models to improve performance
   b. Output different measures of success and evaluate performance
3. A lot more qualitative questions in this project.

Other measures of Success (or Why is Accuracy not Enough)

- Sometimes we care deeply about the TYPE of errors we make, so much so that we may sacrifice our model's overall accuracy if it means avoiding certain errors
  - What might be an example where we'd go out of our way to avoid a false positive?
    - E.g., Facial Recognition by Police
  - How about false negative?
    - E.g., Medical Diagnostic tests



Hw or Project Questions?

- Have a great weekend!