Project T Final: Data Cleaning

Team .N.A.N.T.S.

Visualizing Data with Matplotlib/Seaborn [Optional Review Topic]

Motivation

- Condense large datasets into human comprehensible images
- Sanity check data and model
- Identify outliers
- Explain findings

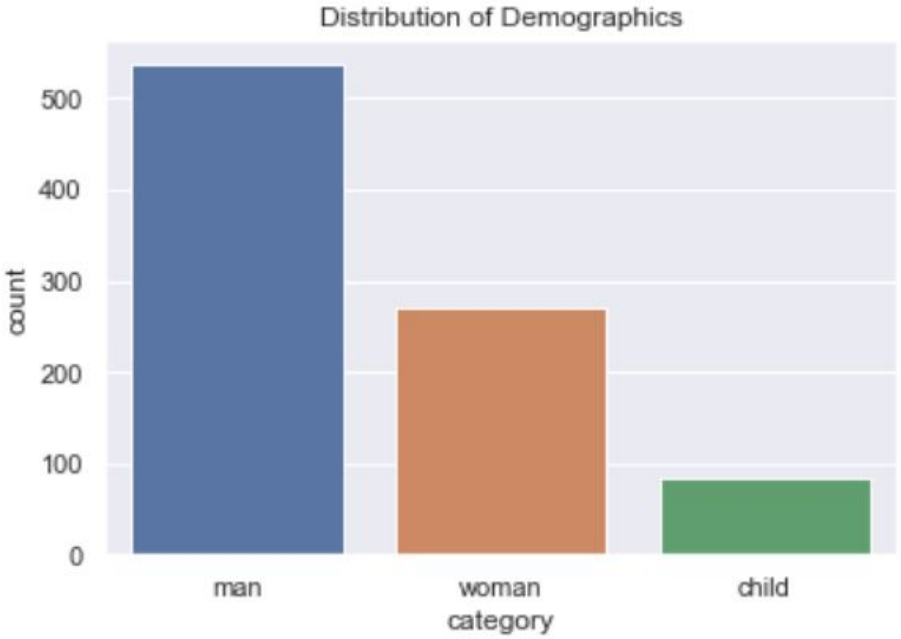
Best Practices when Creating Visualizations

- Add descriptive titles and axis labels.
- Use appropriate scaling so that your data is readable.
- Avoid plotting too much data and cluttering your graph (overplotting).
- Consider if you’re plotting categorical or numerical data.

Single Categorical Variable: Bar Chart

Bar charts visualize the distribution of a single categorical variable. The y-axis can be either counts or percentages.

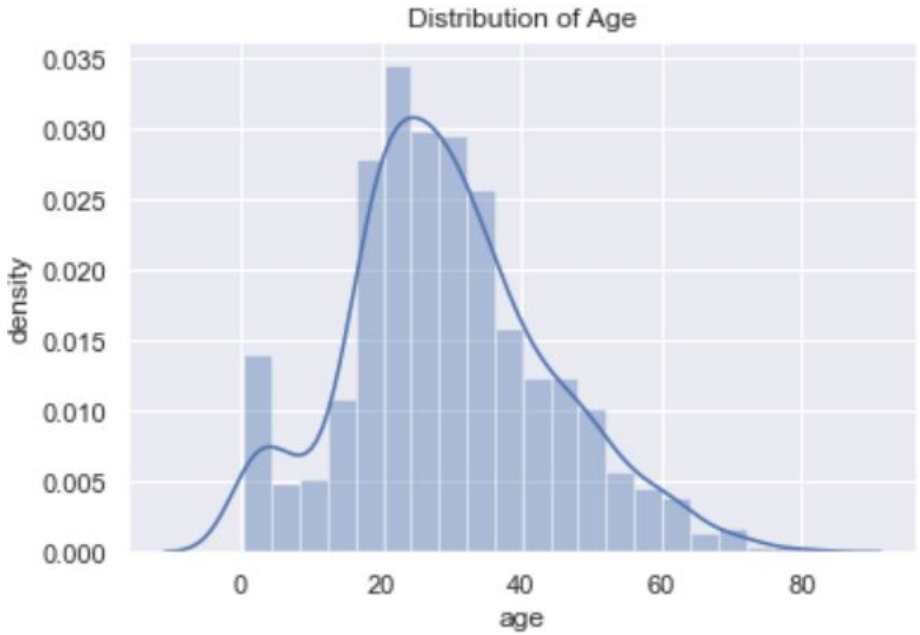
Ex. The bar chart on the right shows the distribution of passengers of the Titanic by gender (except child).



Single Numerical Variable: Histogram

Histograms show the distribution of a single numerical variable. Histograms can use either counts or densities for the y-axis units.

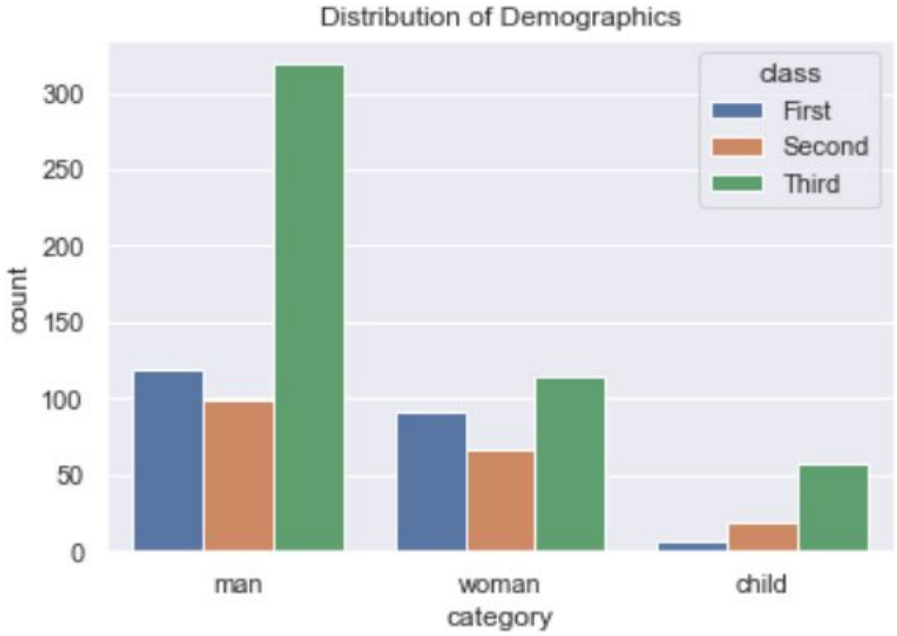
The histogram on the right shows the distribution of the ages of Titanic passengers.



2+ Categorical Variables: Multi-Level Bar Chart

Multi-level bar charts can help visualize multiple categorical variables. Be wary of using this visualization on variables with many labels, as that may compromise readability.

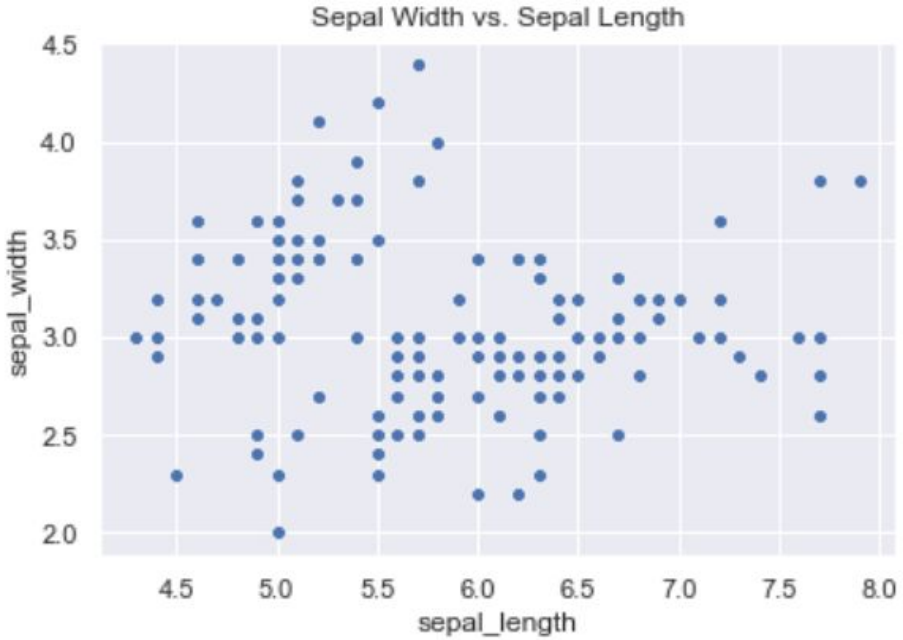
The two-level bar chart on the right shows the distribution of passengers of the Titanic by gender and class.



2 Numerical Variables: Scatter Plot

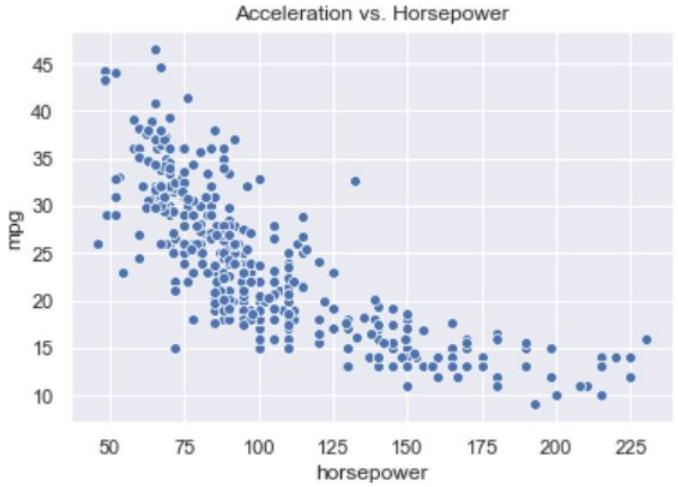
Scatter plots show how two variables are related. While they are powerful visualizations, scatter plots have a few limitations.

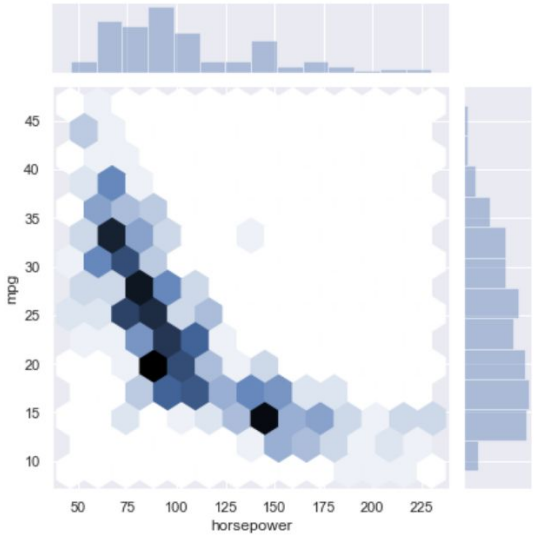
The scatter plot on the right shows how sepal width and sepal length are related in the iris dataset.



2 Numerical Variables: Hex Plot

Hex plots alleviate the problem of overcrowded scatter plots by smoothing out regions. The plots below are a bad scatter plot and its corresponding hex plot:

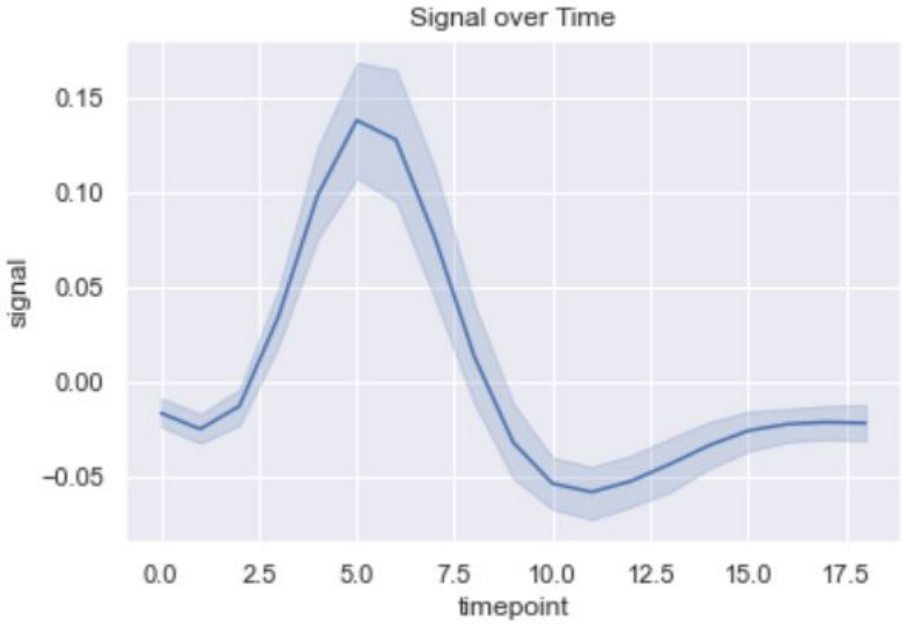




2 Numerical Variables: Line Plot

While scatter plots are useful in the general case when plotting two numerical variables, line plots are more appropriate when one variable does not or rarely repeat, such as in time series data.

The line plot on the right shows signal over time in FMRI time series data.



Additional Notes

- Some visualizations such as scatter plots can include a third variable by using a 3D visualization instead. However, 3D visualizations require a computer to view and sacrifice some readability.
- Numerical visualizations like scatter plots can also incorporate a categorical variable by using different colors (eg. blue points indicate category 1, orange points indicate category 2, etc.).

Normalizing and Standardizing Data

Motivation

- Handle data with features with vastly different ranges of values
- Performed before PCA
- Ex. Seaborn’s mpg dataset has features with different orders of magnitude

horsepower	weight	acceleration
130.0	3504	12.0
165.0	3693	11.5
150.0	3436	11.0
150.0	3433	12.0
140.0	3449	10.5

When to Use Standardization and Normalization

- Algorithms such as SVMs, regularized Regression, k-means clustering and NNs, stochastic gradient descent (SGD)
- In general, algorithms that need a similarity metric between samples which are sensitive to transformations

Normalization

Normalizing a feature maps its values to the range [0, 1]; in other words, fixing the min and max to 0 and 1 respectively. For a data point x_i , the following equation performs the normalization:

$$x_{i,normalized} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

Normalization Example

This slide shows how the “horsepower” and “weight” features of the mpg dataset are normalized. Notice that even though the features have different min and max values, they both end up with 0 and 1 as the new min and max after normalization.

```
mpg["horsepower"].describe()
```

```
count    392.000000
mean     104.469388
std       38.491160
min       46.000000
25%       75.000000
50%       93.500000
75%      126.000000
max      230.000000
```

```
mpg["horsepower_normalized"].describe()
```

```
count    392.000000
mean       0.317768
std        0.209191
min        0.000000
25%        0.157609
50%        0.258152
75%        0.434783
max        1.000000
```

Horsepower

```
mpg["weight"].describe()
```

```
count     398.000000
mean     2970.424623
std       846.841774
min      1613.000000
25%      2223.750000
50%      2803.500000
75%      3608.000000
max      5140.000000
```

```
mpg["weight_normalized"].describe()
```

```
count    398.000000
mean       0.384867
std        0.240103
min        0.000000
25%        0.173164
50%        0.337539
75%        0.565637
max        1.000000
```

Weight

Standardization

Standardization fixes the mean and standard deviation of the feature to be 0 and 1 respectively (note that this does NOT mean exactly a unit gaussian. Given a data point x_i , we have the following equation:

$$x_{i,standardized} = \frac{x_i - x_{mean}}{x_{std}}$$

Standardization Example

This slide shows how the “horsepower” and “weight” features of the mpg dataset are standardized. Notice that the new mean is roughly 0 (numerical imprecision) and the new standard deviation is 1. Additionally, the maximum and minimum describe how much the furthest points vary from the mean.

```
mpg["horsepower"].describe()
```

```
count    392.000000
mean     104.469388
std       38.491160
min       46.000000
25%       75.000000
50%       93.500000
75%      126.000000
max      230.000000
```

```
mpg["horsepower_standardized"].describe()
```

```
count    3.920000e+02
mean    -3.231542e-16
std      1.000000e+00
min     -1.519034e+00
25%     -7.656144e-01
50%     -2.849846e-01
75%      5.593651e-01
max      3.261284e+00
```

Horsepower

```
mpg["weight"].describe()
```

```
count     398.000000
mean     2970.424623
std       846.841774
min      1613.000000
25%      2223.750000
50%      2803.500000
75%      3608.000000
max      5140.000000
```

```
mpg["weight_standardized"].describe()
```

```
count    3.980000e+02
mean    -9.637740e-17
std      1.000000e+00
min     -1.602926e+00
25%     -8.817168e-01
50%     -1.971143e-01
75%      7.528861e-01
max      2.561961e+00
```

Weight

Additional Notes

- You cannot both standardize and normalize a feature; this is because normalizing transforms the feature's range to [0, 1], so the new mean of the feature will be larger than 0 (except in the degenerate case when all values are the same in a feature). This means you must pick one or the other.
- There exists a version of standardization that is more robust to outliers by using median and interquartile range instead of mean and standard deviation. You can read more about sklearn's robust scaler here.

One-Hot Encoding Categorical Data

Motivation

Generally, most models cannot train on a categorical variable, instead requiring numbers rather than labels. One-hot encoding transforms categorical variables into a more computer useable format.

Naive Approach: Enumeration

Replace a set of string labels {label 1, label 2, ..., label n}, with indices {1, 2, ..., n}.

Ex. passenger class from Titanic dataset:

String labels:

class
Third
First
Third
First
Third

Enumerated labels:

pclass
3
1
3
1
3

Problems with Enumeration

Enumeration adds nonsensical mathematical constraints (eg. mapping “First Class” to 1 and “Third Class” to 3 in the Titanic dataset implies that 3 * “First Class” is equal to “Third Class”, or “Second Class” + 1 is equal to “Third Class”).

One-Hot Encoding Technique

Given n unique labels, map label i to a list of n booleans, in which boolean i is the only one marked True.

Ex.

Label 1: [1, 0,..., 0]

Label 2: [0, 1, 0, ... , 0]

Label n: [0, ..., 0, 1]

One-Hot Encoding on the Titanic Dataset

Ex. passenger class of the Titanic dataset:

String labels:

class
Third
First
Third
First
Third

One-hot labels:

class_First	class_Second	class_Third
0	0	1
1	0	0
0	0	1
1	0	0
0	0	1

Additional Notes

- Not all “numerical” data is quantitative. In addition to the previously mentioned enumeration, where a number serves as a proxy for a class, data such as zip codes or phone numbers where mathematical relationships are unclear can also be encoded.
- Can prune away rare labels by merging them together as “Other”.

Processing Text Fields with Regular Expressions

Motivation

- Clean poorly formatted strings, unwanted punctuation
- Extracting useful features from raw text
- Particularly useful in natural language processing NLP

What is Regex?

- Programming tool for performing string manipulations (ex. extracting features from text, replacing substrings)
- Can create a set of characters, or a **pattern**, that can match substrings in other input strings

Common Regex Functions

Regex is commonly imported as “re” in Python. Assuming this is true:

- *re.search(pattern, text)*: Detects whether regular expression pattern is present in the given text.
- *re.sub(pattern, substitute, input)*: Substitutes substrings in input text that match the given pattern with the substitute substring.
- *re.findall(pattern, string)*: Returns a list of all pattern matches in a string.
- *re.compile(pattern)*: Stores the regex pattern in cache memory for faster searches, which is useful if a pattern is reused many times.

Literals

- Straightforward identical pattern
- Ex. "https://eecs.berkeley.edu/" will match exactly the string "https://eecs.berkeley.edu/", and will fully match with no other string
- Partial matches (Ex. for the string "https://eecs.berkeley.edu/news", the trailing “/news” will not be matched)

Character Sets

Commonly used sets:

- [xyz]: Matches exactly the characters “x”, “y”, or “z”
- [a-z]: Matches any lowercase alphabetic character
- [A-Z]: Matches any uppercase alphabetic character
- [a-zA-Z]: Matches any alphabetic character
- [0-9]: Matches any digit between 0 and 9

Can be inverted by using the ^ operator. (Ex. [^0-9] matches all non-numeric characters)

Meta Sequences

Commonly used sets may also have a corresponding meta sequence.

Ex. [0-9] is also represented by \d, \s represents all whitespace characters

Quantifiers and Wildcards

In addition to the previously mentioned character sets and meta sequences, a period (.) is a wildcard matches any single character.

Character sets, meta sequences, and wildcards only match single characters. Quantifiers can specify how many times they need to match:

- *: The preceding character needs to match 0 or more times
- +: The preceding character needs to match 1 or more times
- ?: The preceding character matches 0 or 1 times
- {m, n}: The preceding character matches between m and n times

Basic Regex Example

Task: Extracting a float value from an uncleaned dollar string (eg. “$1,200,689.84 USD").

```
float(re.sub(r"[^\d.]", "", "$1,200,689.84 USD"))
```

```
1200689.84
```

The pattern first finds all characters that are **not** (^) numeric (\d) or periods (.), and replaces them with empty strings. This removes the dollar sign, commas, and suffix of the string, creating a valid string that can be converted into a float.

Additional Notes

- If you want to specify a special character such as a plus sign or period as a literal that's not in a character set, you need to escape the character with backslash.
- The pipe operator (|) specifies an or, which may be useful for specifying different possible characters/substrings at a location in the pattern.
- A great site to debug or sanity check your regex is https://regex101.com/, which highlights exactly what parts a pattern matches in a given input string and pattern, as well as explaining why the match exists.

Handling Imbalanced Class Distributions

Motivation

Ex. Given dataset with distribution: 99% class A, 1% class B

- Naive classifier can achieve training accuracy of 99% by simply classifying everything as class A!
- Naive classifier then generalizes very poorly to real world with distribution of 50% class A, 50% class B

Alternatives to Dataset Modification

1. Collecting more data: can be expensive or difficult in practice
2. Use a different evaluation metric:
    a. Naive classifier weighs false positives and false negatives equally
    b. Ex. A receiver operating characteristic (ROC) curve can help weigh the tradeoff between false positives and false negatives

Resampling Datasets

Common methods:

1. Undersampling: Sample a subset of data points from the majority class, so that fewer majority class points are included.
2. Oversampling: Sample data points from the minority class with replacement, so that minority class points have greater representation.
3. Generating synthetic data: Generate new synthetic data for the minority class; the SMOTE method is commonly used. Synthetic data is similar in concept to oversampling, except data points are not exact copies.

A Warning about Resampling

Given real world with 99% class A and 1% class B, it is actually correct to present training dataset with 99% class A and 1% class B.

Presenting training dataset with 50% class A and 50% class B is actually incorrect in this case!

Handling Null/Missing Values

Motivation

- Can skew dataset with large amounts of default inputs
- Ex. Unfilled age data defaults to 0, average age becomes much lower with large amount of missing data

Two Approaches to Handling Null Values

1. Deletion: Remove the offending variables or rows that contain null values.
2. Imputation: Substitute the null values with some kind of filler value.

Method 1: Deletion

Two options:

1. Deleting rows: Deleting rows is appropriate when there's only a small number of null values, and **if the data is missing at random**. If there is a pattern where the data is null, deleting these rows may lose valuable information and produce biased models.
2. Dropping variables: Dropping a variable is appropriate when a large portion of the variable's values are null (eg. optional field in a survey with few responses), and the variable is insignificant.

Method 2: Imputation (Summary Statistic)

Replace all null values in a variable with the variable's mean, median, or mode.

- Mean and median apply to numerical data while mode applies to categorical or discrete numerical data.
- Easy to implement and not computationally taxing.
- May be prone to bias and takes no advantage of relationships between other variables.

Method 2: Imputation (Linear Regression)

Use a correlation matrix to select good predictors of the variable to impute and train a linear regression model to predict the null values.

- Model may overfit
- Relies on the strong assumption that a linear relationship exists.

Method 2: Imputation (K-Nearest Neighbors)

A kNN model is trained on the dataset, where the imputed variable are what the model predicts. A distance metric between data points is decided, and the model uses the present values of the nearest neighbors to impute the null value.

- KNN model may also overfit.
- Works around the linear relationship assumption linear regression imposed.

Additional Notes

- Categorical variables: can mark null values as “Other” category instead of imputing or deleting.
- With all techniques listed, it is important to consider if the data is missing at random or systematically. Systematically missing data is much more difficult to deal with, since it may be indicative of some unseen, underlying pattern.

Removing Outliers (without OMP)

Motivation

Outliers skew the results of models trained on the data, since the model may try to fit to outliers while sacrificing generalizability.

Defining Outliers

Three common definitions:

1. Data point violates constraints defined by domain knowledge of dataset
2. Data point is visually separated from other points on a graph
3. Data point fails a statistical test or rule

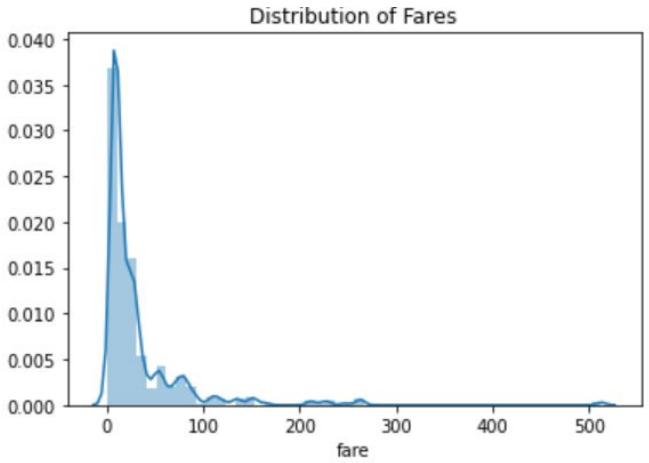
Method 1: Data Point Violates “Common Sense”

Ex. Given age data, a value such as 280 is unrealistic and can be labeled an outlier.

- Introduces subjective bias by introducing curator’s personal notion of “common sense”
- “Common sense” may not exist for more complex real world instances

Method 2: Data Point is Visually Separated on Plot

Ex. distribution of passenger fares in the Titanic dataset:



- Also introduces subjective bias
- Also can have ambiguity
  - While you could reasonably assume the points above 500 are outliers, what about the points between 200 and 300?

Method 3: Using Statistical Tests

Common tests: Peirce's Criterion, Chauvenet's Criterion, IQR rule

Ex. the IQR rule applied to Titanic passenger fares:

```
accepted_range = (titanic["fare"].quantile(0.25) - 1.5 * IQR, titanic["fare"].quantile(0.75) + 1.5 * IQR)
accepted_range
```

```
(-92.724, 131.6344)
```

The IQR rule states that any fare lower than the q1 - 1.5 * IQR or larger than q3 + 1.5 * IQR is an outlier.

- More objective than previous tests
- Still relies on data scientists to make appropriate choice of test to apply

Additional Notes

- It is not always advantageous to remove outliers. If a work is very critical, such as studying the relationship between temperatures and airplane engine failures, removing outliers would be inappropriate.
- Make sure that outliers removed are sparse; if there are many outliers with similar values, that may hint at some underlying true pattern in the data.
- In addition to the methods we've listed, there exists another way to determine outliers with Orthogonal Matching Pursuit (OMP) that is out of scope for this project.

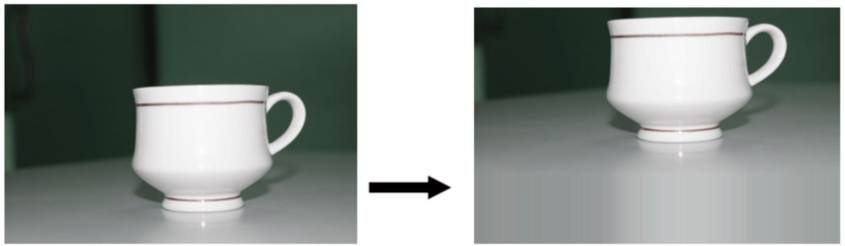
Image Data Augmentation

Motivation

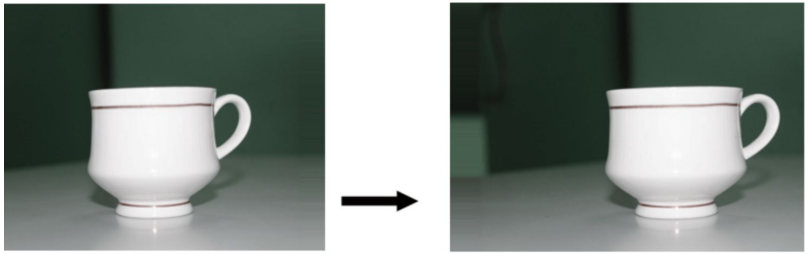
- Increase the effective size of our training dataset by introducing variations of existing samples
- Allow trained model to become more robust to the test and deploy environment

Horizontal and Vertical Shifting

Shifting involves moving the focus of the image up or down, left or right by some number of pixels.



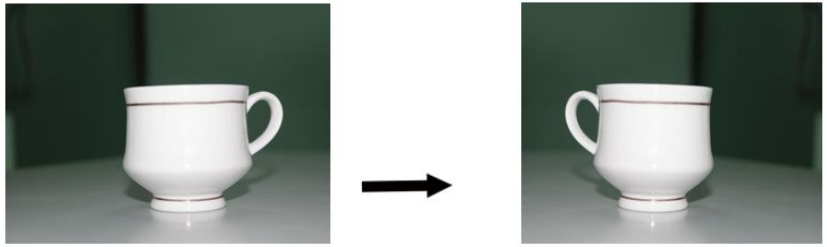
Ex. Vertical shift



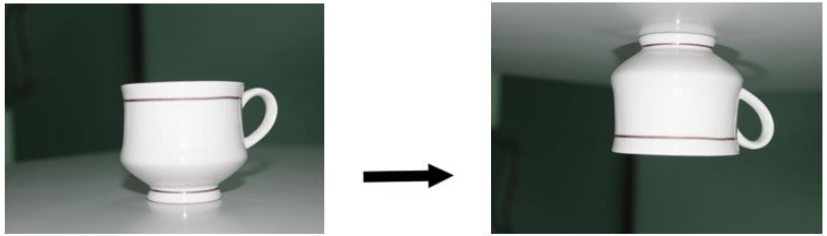
Ex. Horizontal shift

Horizontal and Vertical Flipping

Flipping reflects the image across either axis.



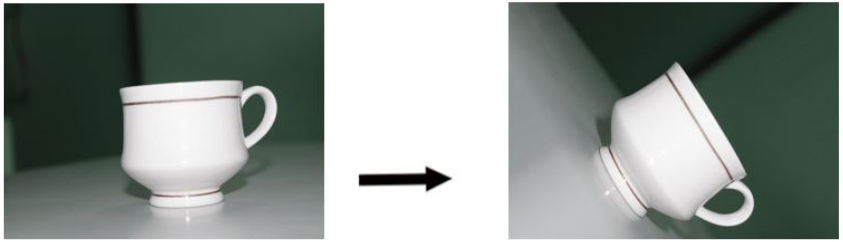
Ex. Horizontal flip



Ex. Vertical flip

Random Rotating

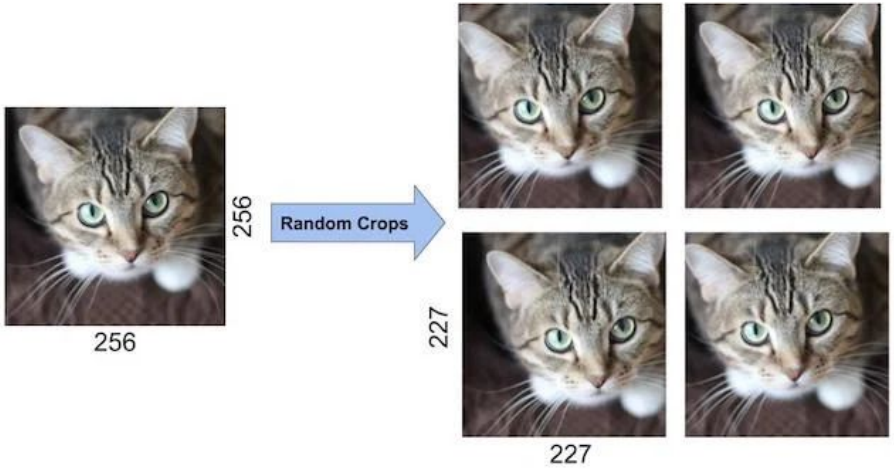
Rotation involves rotating the image at a random angle



Ex. Random rotation

Random Cropping

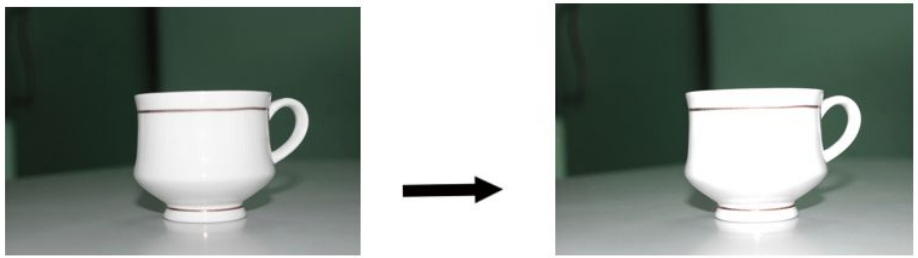
Cropping involves taking a smaller random crop of the original image.



Ex. Random cropping

Random Brightness

Random brightness involves altering the intensities of the RGB channels in the training images. The exact algorithm involves PCA and is described in the Alexnet paper (2012).



Ex. Random brightness

Standard practices (Common transforms)

Standard practice for image data augmentation is as follows:

1. Scale resolution down to 256 x 256
2. Take a random 224 x 224 crop (alternatively, consistently take crops of corners and center)
3. Generate horizontal flip
4. Perform brightness augmentation (optional)

Standard practices (How it is applied)

Augmentation is usually performed at the dataloader step.

1. Training pipeline requests a batch of images during an epoch
2. Dataloader grabs a batch of raw images from the saved folder of training / val data
3. Dataloader applies transforms (perhaps with random factors) onto batch of raw images
4. Dataloader sends transformed images as input into model

Data Decorrelation and Whitening

Correlation in Data

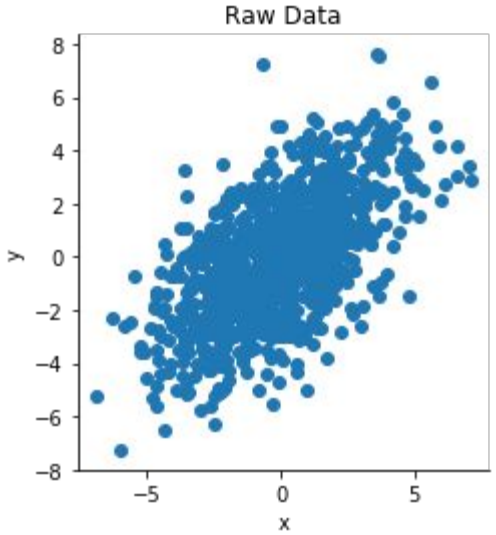
- The concept of a correlation matrix for our data matrix should be familiar since this was part of teaching SVD and PCA in EE16B
- In general, we would like for our data to have uncorrelated features:
  - Uncorrelated features makes the optimization of Neural Networks more efficient
  - Correlations in data potentially means that there is redundant information which means that we can compress our data
- Our discussion of decorrelation and later whitening will depend heavily on the PCA/SVD topics discussed in class, so if you are unfamiliar, we recommend brushing up with those topics

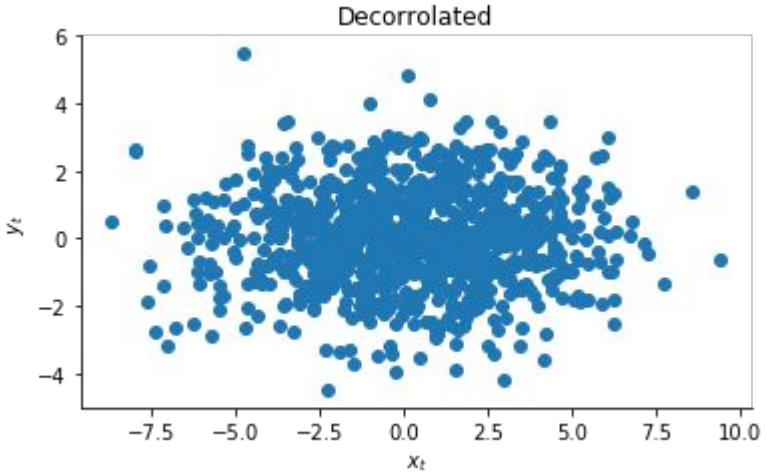
Decorrelating Data

- To decorrelate data, it means to remove the any correlations between different features in the data
- We decorrelate data by zero meaning the features and multiplying by the U matrix that we get from the SVD
- This allows us to get a data matrix that has a diagonal correlation

$$\tilde{X} = U^\top X \implies \tilde{X}^\top \tilde{X} = U^\top U \Lambda U^\top U = \Lambda$$

Decorrolated Data Example





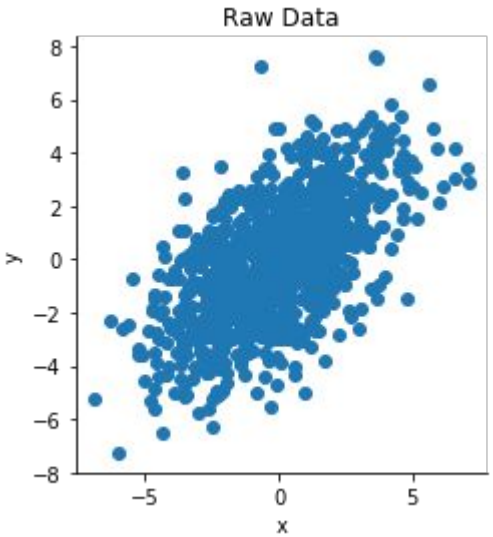
As we can see, after decorrelating the data, the features are now independent of each other

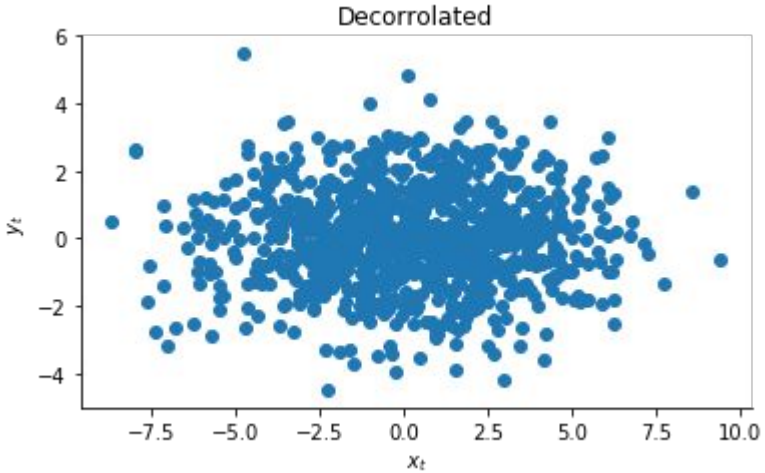
Whitening Data

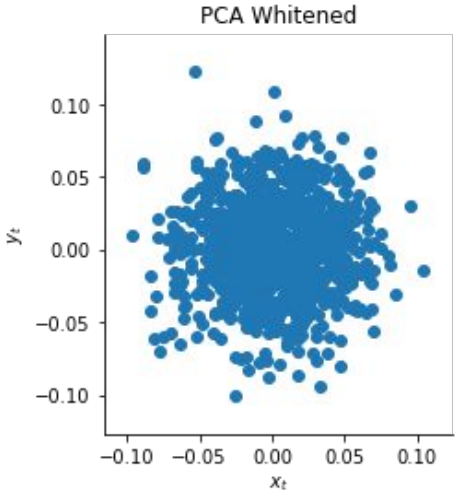
- Whitening is a linear transformation used to both decorrelate different features and set the variances of different features to be identical
  - This names comes from the term “white noise”
- In our case, it means that we wish for the covariance matrix of the data to be the identity matrix rather than just diagonal
  - We can accomplish this by simply scaling each feature by the singular values:
  - This also known as PCA Whitening

$$\tilde{X} = \Lambda^{-1/2} U^\top X \implies \tilde{X}^\top \tilde{X} = U^\top U \Lambda^{-1/2} \Lambda \Lambda^{-1/2} U^\top U = I$$

PCA Whitening example







As we can see, PCA whitening lets us transform the data into a spherical blob of data

ZCA Whitening

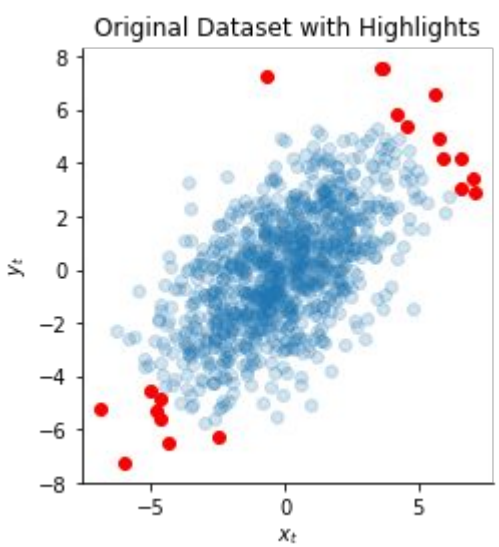
- One thing of note is that the linear transformation by PCA whitening is not unique: any rotation matrix applied on top of the rotation matrix maintain a identity as the covariance
- In Zero-Phase Component Analysis (ZCA) whitening, we decide to choose the matrix U as our rotation matrix:

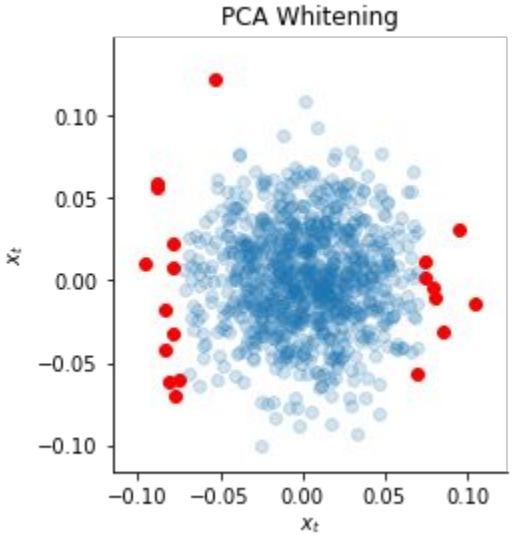
$$\tilde{X} = U\Lambda^{-1/2}U^{\top}X$$

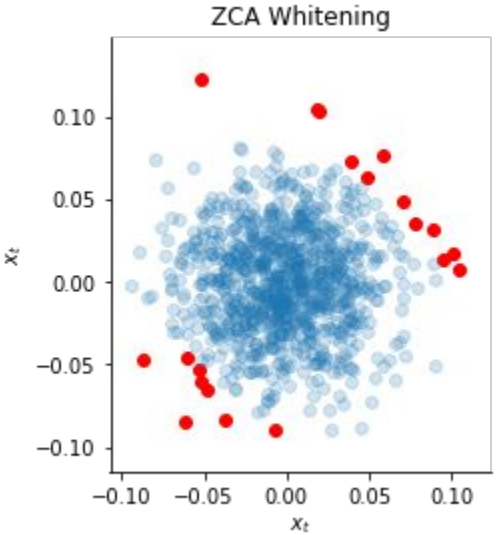
PCA whitening vs. ZCA whitening

- PCA whitening is ideal if your goal is to try and compress the data after applying whitening
- You can always just keep the top k principal components, and you will still have identity as your covariance
- ZCA whitening is ideal if you want the transformed random vector to be as similar as possible to the original vector
    - See source for the exact proof which we will leave out here

PCA vs ZCA Whitening







The orientation of the data is loosely preserved with ZCA

Remarks

- Avoid the common pitfall of not applying the transformations from the training set to the test set when testing
- The effect of whitening can be substantial on neural networks, but it really depends on the specific optimizer
  - Whitening is not commonly used in Convnets, standardizing and zero-mean is more common
  - See this source for more about gradient descent