Data cleaning and feature engineering

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

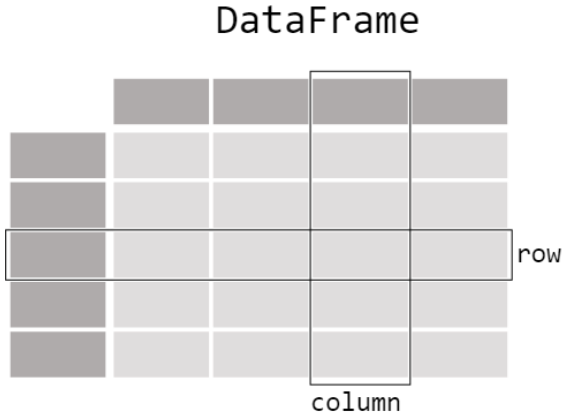
Understand your data

- It's pretty bad practice to treat a dataset as a black box. Before you start training models, you should explore and visualize your data to gain insights about what makes it predictive, which will inform feature engineering and screen for potential issues
  - If your data includes images or natural language text, take a look at a few samples (and their labels) directly
  - If your data contains numerical/categorical features, it's a good idea to plot the histogram of feature values to get a feel for the range of values taken and the frequency of different values
  - Are some samples missing values for some features? If so, you'll need to deal with this when you prepare the data
  - If your task is a classification problem, print the number of instances of each class in your data. Are the classes roughly equally represented? If not, *you will need to account for this imbalance*
  - If your data includes location information, plot it on a map. Do any clear patterns emerge?

Prepare the data

- As you've learned before, models typically don't ingest raw data
- Data preprocessing/preparation aims at making the raw data at hand more amenable to model
  - This includes vectorization, normalization, encoding or handling missing values
  - Many preprocessing techniques are domain-specific (for example, specific to text data or image data)
  - We will focus on tabular data here

This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn . . .



Data cleaning

- Data cleaning is a key part of data science, but it can be deeply frustrating.
  - Why are some of your text fields garbled?
  - What should you do about those missing values?
  - Why aren't your strings/dates formatted correctly?
  - How can you quickly clean up inconsistent/duplicated data entry?
- The importance of cleaning data
  1. Ease of use and reuse: When data is properly organized and normalized it's easier to search, use, and share with others
  2. Consistency: Data science often requires working with more than one dataset, where datasets from different sources need to be joined together. Making sure that each individual data set has common standardization will ensure that the data is still useful when they are all merged into one dataset
  3. Model accuracy/interpretability: Data that has been cleaned improves the models

Handling missing values - Figure out why the data is missing

- For dealing with missing values, you'll need to use your intuition to figure out why the value is missing. One of the most important questions you can ask yourself to help figure this out is this:
  - Is this value missing because it wasn't recorded or because it doesn't exist?
  - If a value is missing because it doesn't exist (like the height of the oldest child of someone who doesn't have any children) then it doesn't make sense to try and guess what it might be. These values you probably do want to keep as `NaN`
  - If a value is missing because it wasn't recorded, then you can try to guess what it might have been based on the other values in that column and row. This is called *imputation*
    - For example, if we form a matrix of the ratings (on a scale from 1 to 5) that n customers have given to the entire Netflix catalog of p movies, then most of the matrix will be missing, since no customer will have seen and rated more than a tiny fraction of the catalog
    - If we can impute the missing values well, then we will have an idea of what each customer will think of movies they have not yet seen

1. Handling missing values

- Sometimes you could just discard the feature entirely, but you don’t necessarily have to
  - If the feature is categorical, it’s safe to create a new category that means “the value is missing.” The model will automatically learn what this implies with respect to the targets.
  - If the feature is numerical, avoid inputting an arbitrary value like "0", because it may create a discontinuity in the latent space formed by your features, making it harder for a model trained on it to generalize. Instead, consider replacing the missing value with the average or median value for the feature in the dataset. You could also train a model to predict the feature value given the values of other features

Handling missing values - Dropping or simple filling

- One option is drop it
  - If you're in a hurry or don't have a reason to figure out why your values are missing, one option you have is to just remove any rows or columns that contain missing values
  - However, this comes at the price of losing data which may be valuable (even though incomplete)
  - You could set a threshold to retain more data
- You can also keep it as it is, but you should fill it with a value
  - Using something like -9999 which is a value out of normal range and feed it into ensemble model or random forest
- Another option is to try and fill it with logical order
  - You could use entry below or previous entry to fill the value
  - This makes a lot of sense for datasets where the observations have some sort of logical order to them

Handling missing values - Imputation

- A better strategy is to impute the missing values, i.e., to infer them from the known part of the data
  - One type of imputation algorithm is univariate, which imputes values in the i-th feature dimension using only non-missing values in that feature dimension
  - By contrast, multivariate imputation algorithms use the entire set of available feature dimensions to estimate the missing values
- If x_{ij} is missing, then we could replace it by the mean of the jth column (using the non-missing entries to compute the mean)
  - You could also fill with other representative value like the medium
  - Categorical variable can be filled with most frequent value or treat it as a separate level
  - Although this is a common and convenient strategy, often we can do better by exploiting the correlation between the variables

Handling missing values - Iterative strategy

- It does so in an iterated round-robin fashion: at each step, a feature column is designated as output y and the other feature columns are treated as inputs X
  - The row contains valid value are treated as training data while the row with missing value are treated as target
  - A regressor is fit on (X, y) for known y. Then, the regressor is used to predict the missing values of y. This is done for each feature in an iterative fashion, and then is repeated for a fix number of imputation rounds. The results of the final imputation round are returned
  - You can pass different regressors for predicting missing feature values

Handling missing values - Nearest neighbor strategy

- The KNN approach provides imputation for filling in missing values using the k-Nearest Neighbors approach
  - By default, a Euclidean distance metric is used to find the nearest neighbors
  - Each missing feature is imputed using values from n nearest observations that have a value for the feature. The feature of the neighbors are averaged uniformly or weighted by distance to each neighbor
  - If a sample has more than one feature missing, then the neighbors for that sample can be different depending on the particular feature being imputed

Handling missing values - Matrix completion

- We can assume that the first M principal component score and loading vectors provide the "best" approximation to the data matrix X
- Now, some of the observations x_{ij} are missing. One can both impute the missing values and solve the principal component problem at the same time

$$\min_{A \in R^{n \times M},\, B \in R^{p \times M}} \{ \sum_{(i,j) \in O} (x_{ij} - \sum_{m=1}^{M} a_{im} b_{jm})^2 \}$$

where O is the set of all observed pairs of indices (i, j), a subset of the possible $n \times p$ pairs

  - We can estimate a missing observation x_{ij} using $x_{ij} = \sum_{m=1}^{M} \hat{a}_{im} \hat{b}_{jm}$
  - We can (approximately) recover the M principal component scores and loadings, as we did when the data were complete

Algorithm 12.1 *Iterative Algorithm for Matrix Completion*

1. Create a complete data matrix $\tilde{\mathbf{X}}$ of dimension $n \times p$ of which the (i, j) element equals

$$\tilde{x}_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in \mathcal{O} \\ \bar{x}_j & \text{if } (i,j) \notin \mathcal{O}, \end{cases}$$

   where $\bar{x}_j$ is the average of the observed values for the jth variable in the incomplete data matrix $\mathbf{X}$. Here, $\mathcal{O}$ indexes the observations that are observed in $\mathbf{X}$.

2. Repeat steps (a)–(c) until the objective (12.14) fails to decrease:

   (a) Solve

$$\underset{\mathbf{A}\in\mathbb{R}^{n\times M},\mathbf{B}\in\mathbb{R}^{p\times M}}{\text{minimize}} \left\{ \sum_{j=1}^{p}\sum_{i=1}^{n}\left(\tilde{x}_{ij} - \sum_{m=1}^{M} a_{im}b_{jm}\right)^2 \right\} \tag{12.13}$$

   by computing the principal components of $\tilde{\mathbf{X}}$.

   (b) For each element $(i, j) \notin \mathcal{O}$, set $\tilde{x}_{ij} \leftarrow \sum_{m=1}^{M} \hat{a}_{im}\hat{b}_{jm}$.

   (c) Compute the objective

$$\sum_{(i,j)\in\mathcal{O}}\left(x_{ij} - \sum_{m=1}^{M}\hat{a}_{im}\hat{b}_{jm}\right)^2. \tag{12.14}$$

3. Return the estimated missing entries $\tilde{x}_{ij}$, $(i, j) \notin \mathcal{O}$.

Duplicate entry

- In addition to missing data, you will often encounter duplicated data in real-world datasets. Fortunately, many packages provides an easy means of detecting and removing duplicate entries
  - You can identifying duplicates and drop them using pandas
- Removing duplicate data is an essential part of almost every data-science project. Duplicate data can change the results of your analyses and give you inaccurate results!

letters	numbers
A	1
B	2
A	1
B	3
B	3

Inconsistent entry

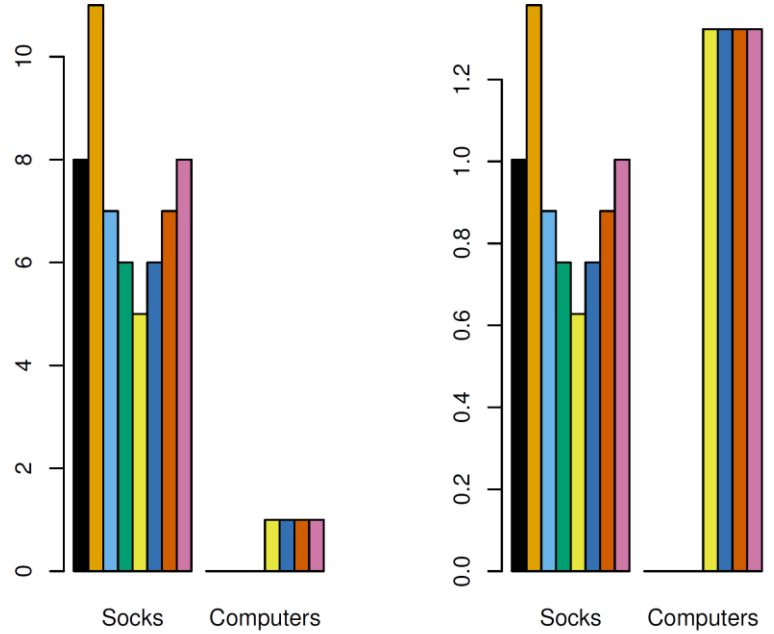
- Data can have inconsistencies in how it's presented
  - This can cause problems in searching for and representing the value, where it's seen within the dataset but is not properly represented in visualizations or query results. Common formatting problems involve resolving whitespace, dates, inconsistent names and data types
  - Resolving formatting issues is typically up to the people who are using the data. For example, standards on how dates and numbers are presented can differ by country
- Fuzzy matching is the process of automatically finding text strings that are very similar to the target string
  - In general, a string is considered "closer" to another one the fewer characters you'd need to change if you were transforming one string into another
  - So "apple" and "snapple" are two changes away from each other (add "s" and "n") while "in" and "on" are one change away (replace "i" with "o")

Character encoding

- Character encodings are specific sets of rules for mapping from raw binary byte strings (that look like this: 0110100001101001) to characters that make up human-readable text (like "hi")
  - There are many different encodings, and if you tried to read in text with a different encoding than the one it was originally written in, you ended up with scrambled text
  - You might also end up with a "unknown" characters. There are what gets printed when there's no mapping between a particular byte and a character in the encoding you're using to read your byte string in
  - Character encoding mismatches are less common today than they used to be, but it's definitely still a problem. There are lots of different character encodings, but the main one you need to know is UTF-8
    - UTF-8 is the standard text encoding. All Python code is in UTF-8 and, ideally, all your data should be as well. It's when things aren't in UTF-8 that you run into trouble

2. Scaling of the variables matters

- An eclectic online retailer sells two items: socks and computers, the same scaling might be undesirable, since
  - Computers are more expensive than socks and so the online retailer may be more interested in encouraging shoppers to buy computers than socks, and a large difference in the number of socks purchased by two shoppers may be less informative about the shoppers' overall shopping preferences than a small difference in the number of computers purchased



Scaling and standardization

- In both cases, you're transforming the values of *numeric variables* so that the transformed data points have specific helpful properties. The difference is that:
- Min-max scaling means that you're transforming your data so that it fits within a specific scale, like 0-100 or 0-1. To scale between $[a, b]$

$$\tilde{x} = a + \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}}$$

- You can use standardization If you expect your data can be described as a normal distribution

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

Nonlinear transform

- There are also other non-linear transformation
  - Sometimes you may want to use logarithm transform for the currency
  - Sometimes you may want to use binning to transform numerical value into categorical one
- In some cases, you'll normalize your data if you're going to be using a machine learning or statistics technique that assumes your data is normally distributed. Some examples of these include linear discriminant analysis (LDA) and Gaussian naive Bayes
  - Power transforms are a family of parametric, monotonic transformations that aim to map data from any distribution to as close to a Gaussian distribution as possible in order to stabilize variance and minimize skewness
    - Box-cox (applied to strictly positive data) and Yeo-Jonson transform
  - Quantile transform provides a non-parametric transformation to map the data to a uniform distribution with values between 0 and 1 or a Gaussian distribution

3. Encoding categorical variable – One hot encoding

- Categorical data can be extremely useful. However, in its original form, it is unrecognizable to most models. We can use different "encoding" techniques
  - *One hot encoding* convert it to dummy variables by produces one feature per category
    - In linear and logistic regression, one hot encoding causes problems with multicollinearity. In such cases, one dummy is omitted (its value can be inferred from the other values)
    - The number of categorical features should be small so that it can be effectively applied

Animal	Target	isCat	isDog	isHamster
Cat	1	1	0	0
Hamster	0	0	0	1
Cat	0	1	0	0
Dog	1	0	1	0
Hamster	0	0	0	1
Cat	1	1	0	0
Dog	0	0	1	0

Encoding categorical variable – Label encoding

- Another popular encoding is *ordinal encoding* or *label encoding*, it transforms each categorical feature to one new feature of integers (0 to number of features-1)
  - This coding suggests an *ordering*. Furthermore, it implies that the difference between cat and dog is the same as between dog and hamster

Animal	Target	Animal_encoded
Cat	1	0
Hamster	0	2
Cat	0	0
Dog	1	1
Hamster	0	2
Cat	1	0
Dog	0	1

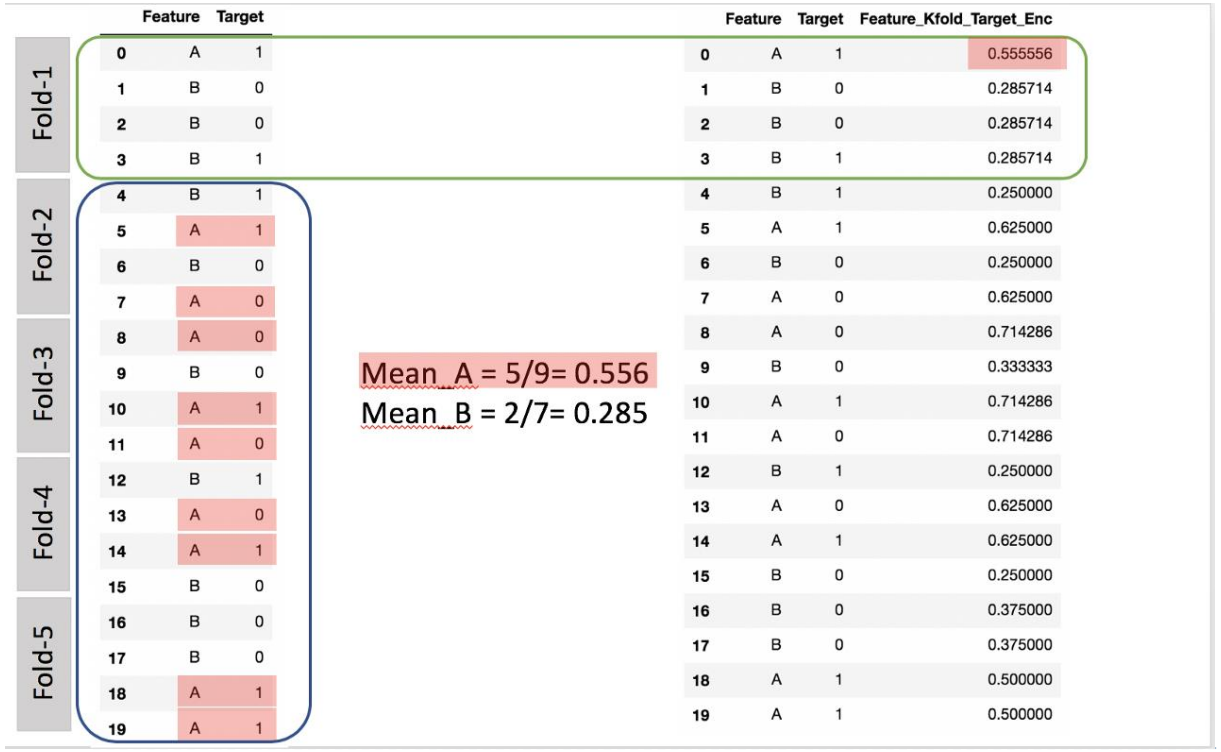
Encoding categorical variable – Target encoding

- In *target encoding* or *mean encoding*, it that replaces a feature's categories with some number derived from the target
  - Group the data by each category and count the number of occurrences of each target. Calculate the average of target given each specific category and add to new column
  - A target encoding derives numbers for the categories using the feature's most important property: its relationship with the target

Animal	Target	Animal_encoded
Cat	1	0.67
Hamster	0	0.50
Cat	0	0.67
Dog	0	0.00
Hamster	1	0.50
Cat	1	0.67
Dog	0	0.00

Encoding categorical variable – Target encoding

- When a category only occurs a few times in the dataset, any statistics calculated on its group are unlikely to be very accurate and may leak the target
  - To avoid target leak and overfitting, target encoding need to be trained on an independent "encoding" split. You can use cross-validation in practice



	Feature	Target
0	A	1
1	B	0
2	B	0
3	B	1
4	B	1
5	A	1
6	B	0
7	A	0
8	A	0
9	B	0
10	A	1
11	A	0
12	B	1
13	A	0
14	A	1
15	B	0
16	B	0
17	B	0
18	A	1
19	A	1

	Feature	Target	Feature_Kfold_Target_Enc
0	A	1	0.555556
1	B	0	0.285714
2	B	0	0.285714
3	B	1	0.285714
4	B	1	0.250000
5	A	1	0.625000
6	B	0	0.250000
7	A	0	0.625000
8	A	0	0.714286
9	B	0	0.333333
10	A	1	0.714286
11	A	0	0.714286
12	B	1	0.250000
13	A	0	0.625000
14	A	1	0.625000
15	B	0	0.250000
16	B	0	0.375000
17	B	0	0.375000
18	A	1	0.500000
19	A	1	0.500000

	Feature	Target	Feature_Kfold_Target_Enc
0	A	1	0.555556
1	B	0	0.285714
2	B	0	0.285714
3	B	1	0.285714
4	B	1	0.250000
5	A	1	0.625000
6	B	0	0.250000
7	A	0	0.625000
8	A	0	0.714286
9	B	0	0.333333
10	A	1	0.714286
11	A	0	0.714286
12	B	1	0.250000
13	A	0	0.625000
14	A	1	0.625000
15	B	0	0.250000
16	B	0	0.375000
17	B	0	0.375000
18	A	1	0.500000
19	A	1	0.500000

Mean_A = (.5556+.625+.625+0.714286+0.714286+0.714286+0.625+0.625+0.5+0.5)/10
Mean_A = 0.61981

	Feature
0	B
1	B
2	B
3	A
4	A

	Feature	Feature_Kfold_Target_Enc
0	B	0.294048
1	B	0.294048
2	B	0.294048
3	A	0.619841
4	A	0.619841

https://medium.com/@pouryaayria/k-fold-target-encoding-dfe9a594874b

Encoding categorical variable – Target encoding

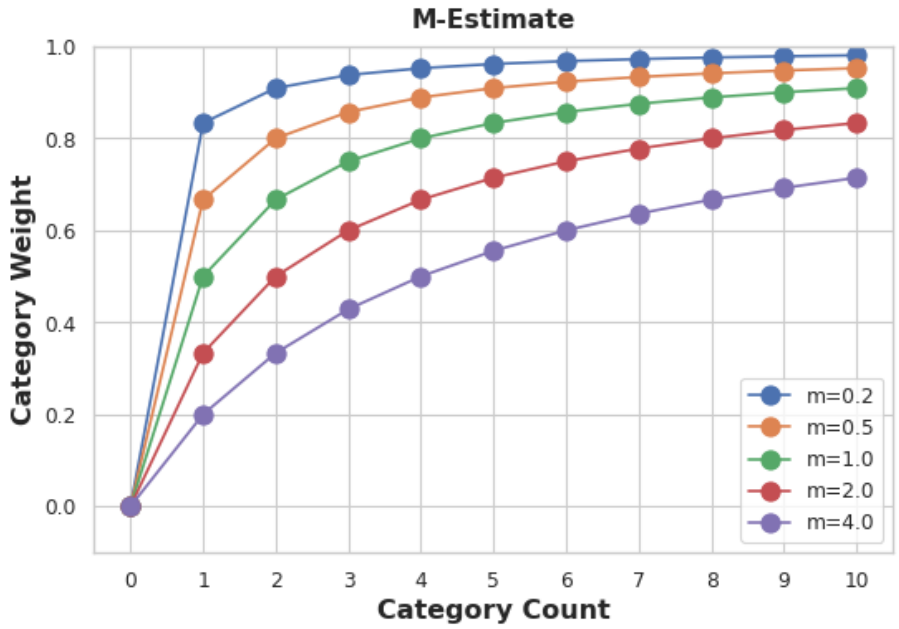
- Another solution to these problems is to add *smoothing*. The idea is to blend the *in-category* average with the *overall* average. Rare categories get less weight on their category average and missing categories just get the overall average

$$encoding = weight \times in\_category + (1 - weight) \times overall$$

- An easy way to determine the value for weight is to compute an m-estimate:

$$weight = n / (n + m)$$

where n is the total number of times that category occurs in the data. The parameter m determines the "smoothing factor". Larger values of m put more weight on the overall estimate

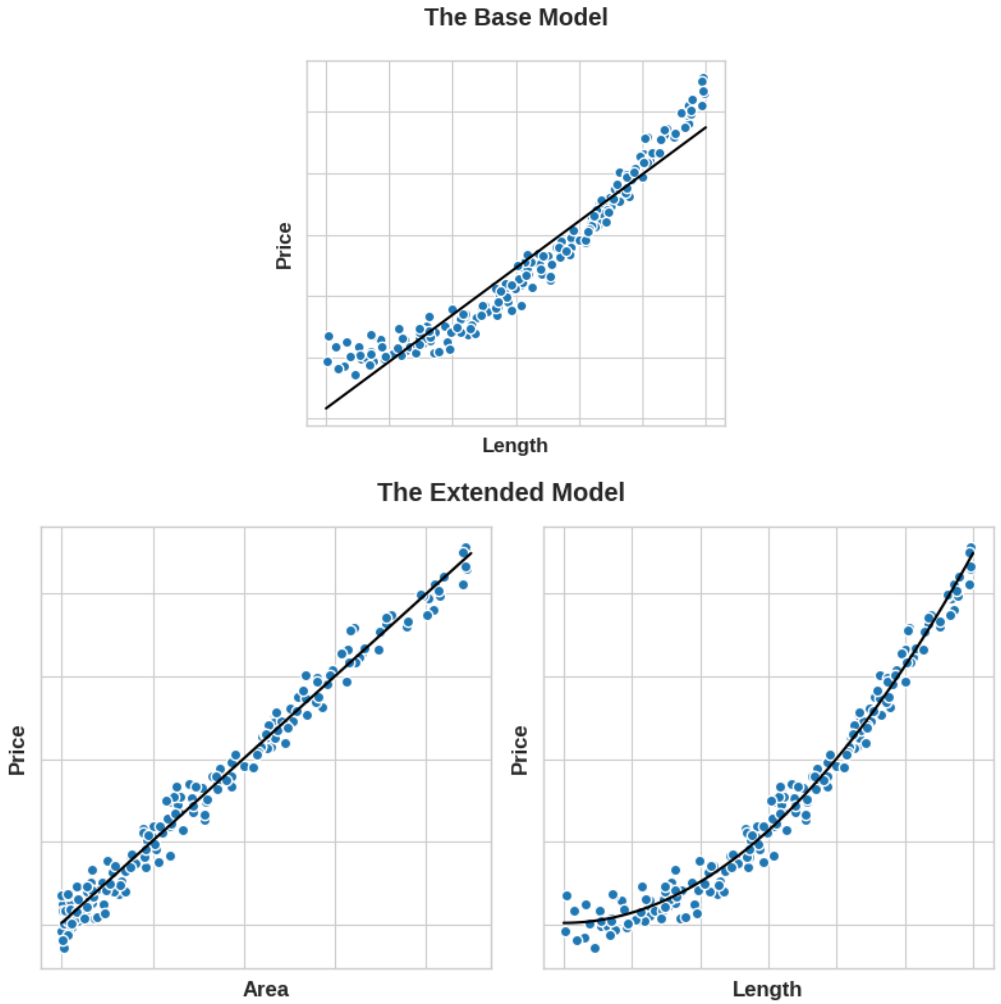


What is Feature engineering?

- One of the most important steps on the way to building a great machine learning model is feature engineering. You might perform it to:
  - Improve a model's predictive performance
  - Reduce computational or data needs
  - Improve interpretability of the results
- The goal of feature engineering is simply to make your data better suited to the problem at hand
  - Consider "apparent temperature" measures like the heat index and the wind chill. These quantities attempt to measure the perceived temperature to humans based on air temperature, humidity, and wind speed, things which we can measure directly
  - You could think of an apparent temperature as the result of a kind of feature engineering, an attempt to make the observed data more relevant to what we actually care about: how it actually feels outside!

What is Feature engineering?

- For a feature to be useful, it must have a relationship to the target that your model is able to learn
  - Linear models, for instance, are able to learn linear relationships. So, when using a linear model, your goal is to transform the features to make their relationship to the target linear
    - Say you were trying to predict the Price of square plots of land from the Length of one side. Fitting a linear model directly to Length gives poor results: the relationship is not linear
    - If we square the Length feature to get 'Area', however, we create a linear relationship. Adding Area to the feature set means this linear model can now fit a parabola. Squaring a feature, in other words, gave the linear model the ability to fit squared features



Feature engineering

- A great first step is to construct a ranking with a feature utility metric, a function measuring associations between a feature and the target
  - Then you can choose a smaller set of the most useful features to develop initially and have more confidence that your time will be well spent
  - We will cover feature selection in next lecture
- Once you've identified a set of features with some potential, it's time to start developing them. Some tips are below
  - Understand the features. Refer to your dataset's data documentation, if available
  - Research the problem domain to acquire domain knowledge. If your problem is predicting house prices, do some research on real-estate for instance. Wikipedia can be a good starting point, but books and journal articles will often have the best information
  - Study previous work. Solution write-ups from past Kaggle competitions are a great resource

Tips on discovering/creating new features

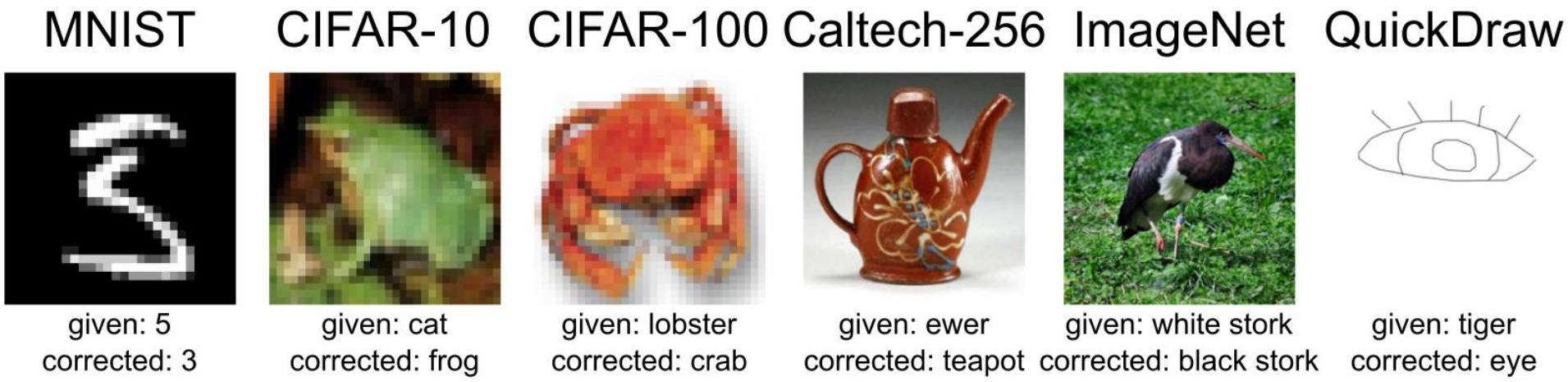
- Use data visualization. Visualization can reveal pathologies in the distribution of a feature or complicated relationships that could be simplified. Be sure to visualize your dataset as you work through the feature engineering process
  - You can identify promising transform by Exploratory Data Analysis (EDA)
  - Typical transformation
    - Interaction between features
    - You can apply arithmetic operations to columns (Ratio, Log, Square…)
    - You can compute statistics for each row like the number of missing value, number of zeros, mean, max, min…

Tips on discovering/creating new features

- It's good to keep in mind your model's own strengths and weaknesses when creating features. Here are some guidelines:
  - Linear models learn sums and differences naturally, but can't learn anything more complex
  - Ratios seem to be difficult for most models to learn. Ratio combinations often lead to some easy performance gains
  - Linear models and neural nets generally do better with normalized features. Neural nets especially need features scaled to values not too far from 0. Tree-based models (like random forests and XGBoost) can sometimes benefit from normalization, but usually much less so
  - Tree models can learn to approximate almost any combination of features, but when a combination is especially important they can still benefit from having it explicitly created, especially when data is limited
  - Counts are especially helpful for tree models, since these models don't have a natural way of aggregating information across many features at once

Other methods for improving your dataset

- cleanlab automatically finds and fixes label issues in your ML datasets
  - It can reduces manual work needed to fix data errors and helps train reliable ML models on noisy real-world datasets
  - The key idea is confidence learning and rank pruning



https://github.com/cleanlab/cleanlab

Conclusion

- It’s pretty bad practice to treat a dataset as a black box. Before you start training models, you should explore and visualize your data to gain insights about what makes it predictive
- We can then screen for potential issues and perform data cleaning
- This will also inform and discover useful feature engineering

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 1

[2] https://madewithml.com/courses/mlops/preprocessing/

[3] https://github.com/microsoft/Data-Science-For-Beginners/blob/main/2-Working-With-Data/08-data-preparation/README.md

[4] https://www.kaggle.com/learn/feature-engineering

[5] https://www.kaggle.com/learn/data-cleaning

[6] An Introduction to Statistical Learning, second edition

Appendix

Resources and libraries

- Data cleaning and data centric AI
  - https://github.com/HazyResearch/data-centric-ai
  - https://github.com/cleanlab/cleanlab
- Data imputation
  - https://github.com/iskandr/fancyimpute
- Exploratory data analysis
  - https://www.kaggle.com/learn/data-visualization
  - https://madewithml.com/courses/mlops/exploratory-data-analysis/
  - pandas-profiling
  - dataprep
  - lux
  - Pycaret

Resources and libraries

- Automatic feature engineering
  - https://github.com/EthicalML/awesome-production-machine-learning#feature-engineering-automation
  - https://github.com/nccr-itmo/FEDOT
  - https://github.com/alteryx/featuretools
- Imbalance data
  - https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data
- Outlier and Anomaly
  - https://github.com/EthicalML/awesome-production-machine-learning#outlier-and-anomaly-detection

Resources and libraries

- Feature store
  - https://github.com/EthicalML/awesome-production-machine-learning#feature-stores
- Features engineering for images
  - https://scikit-learn.org/stable/modules/feature_extraction.html#image-feature-extraction
  - https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html
- Features engineering for natural language
  - https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

Exploratory Data Analysis (EDA) and Data Mining

- The field of exploratory data analysis was established with Tukey's 1977 now-classic book *Exploratory Data Analysis* [Tukey-1977]. Tukey presented simple plots (e.g., boxplots, scatterplots) that, along with summary statistics (mean, median, quantiles, etc.), help paint a picture of a data set.
  - It is important to understand what you can do before you learn to measure how well you seem to have done it
  - Allow the data to speak for themselves before standard assumptions or formal modeling
  - The greatest value of a picture is when it forces us to notice what we never expected to see

https://en.wikipedia.org/wiki/John_Tukey

Visualization

- Seaborn combines simple statistical fits with plotting on pandas dataframes

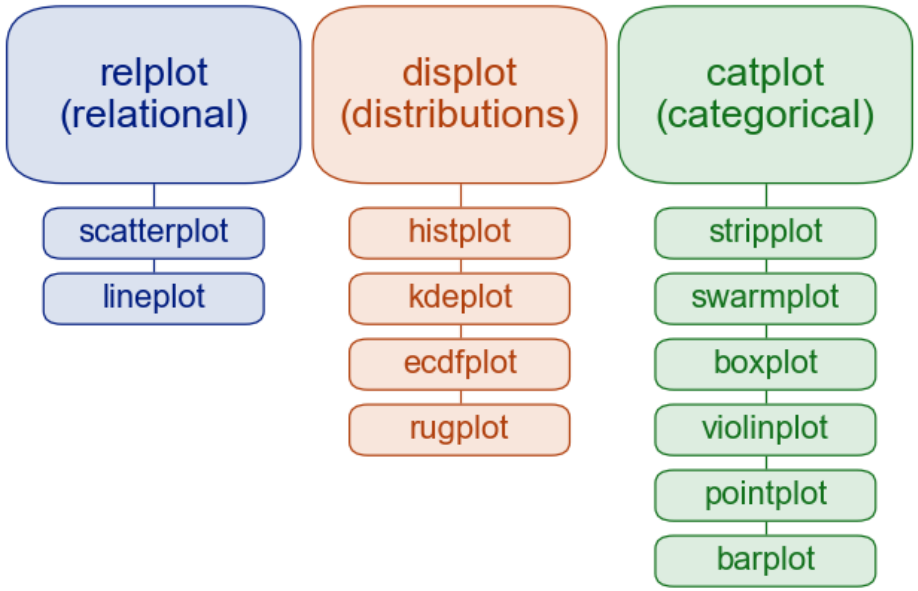


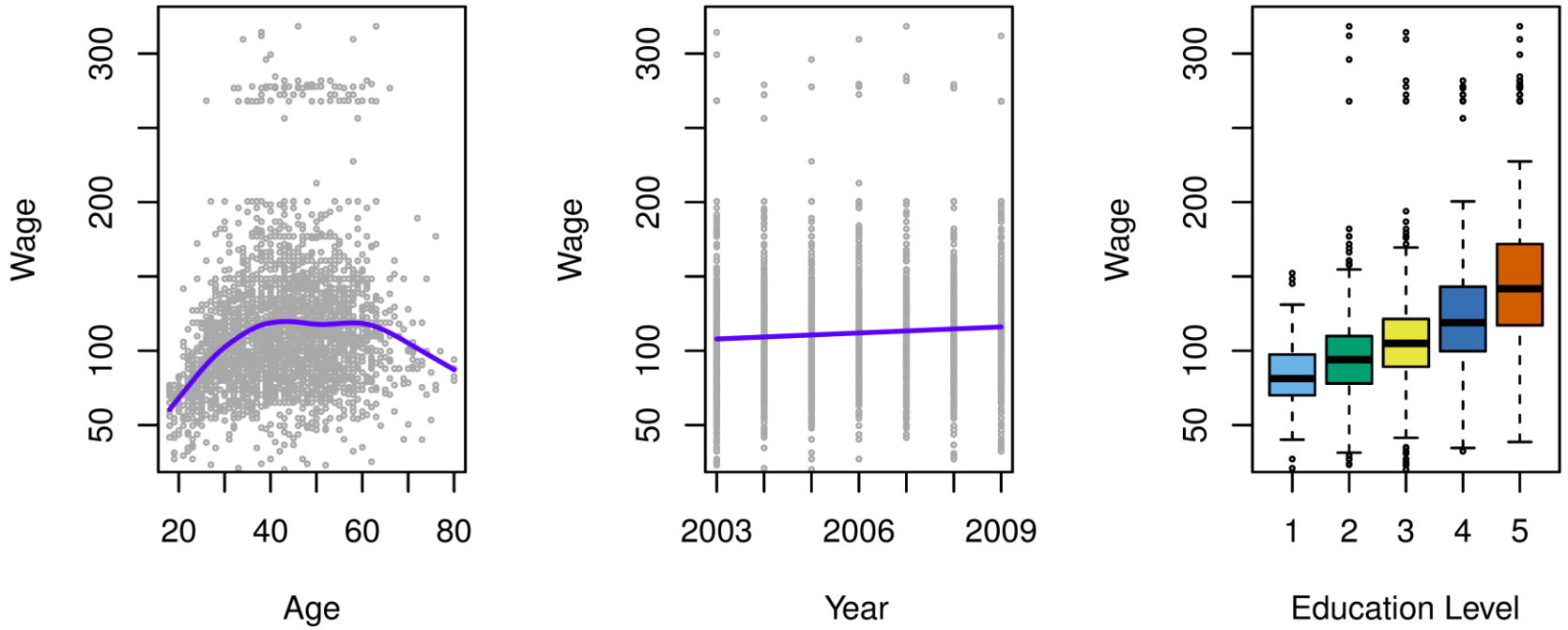
figure-level function

axes-level function

- Multiple plot - joinplot and pairplot
- Regression plot – lmplot, regplot and residplot
- Matrix plot – heatmap and clusterplot

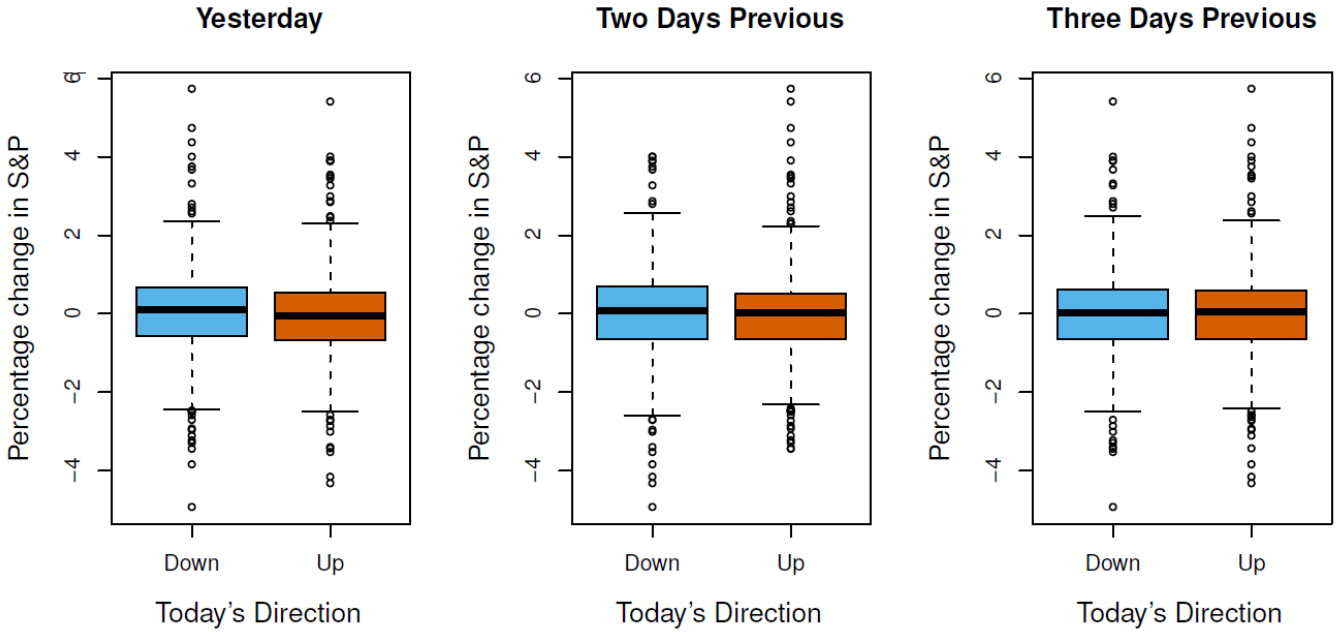
Wage data

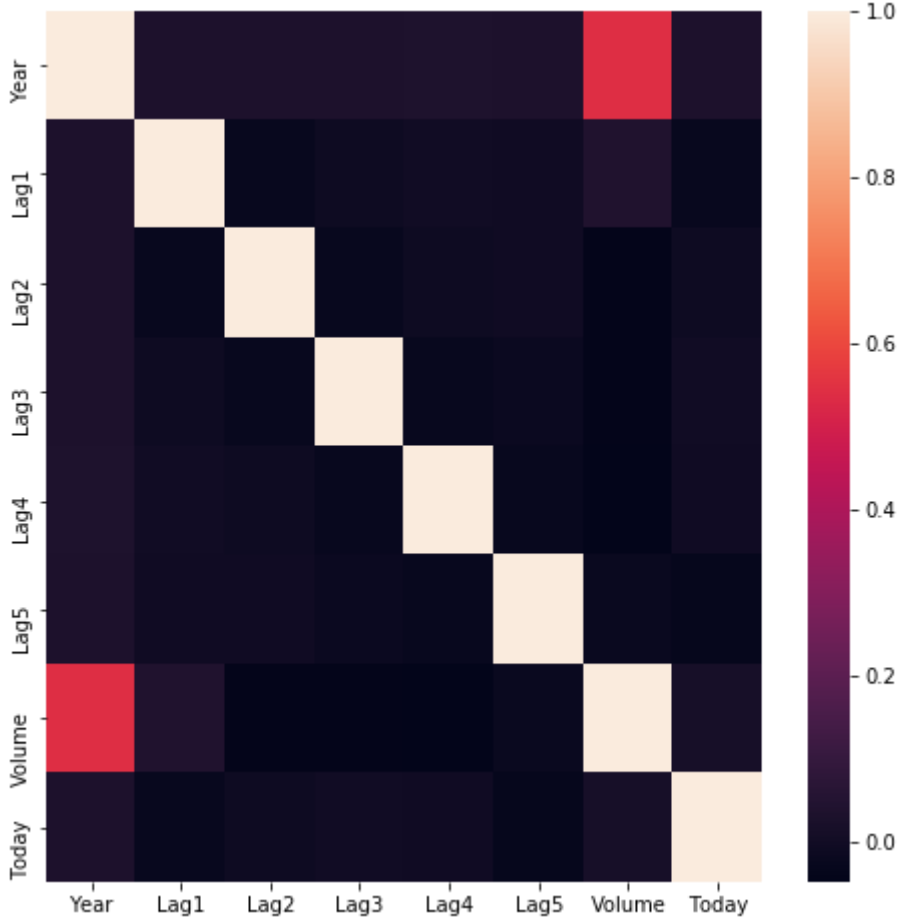
- Dataset from https://www.statlearning.com/
- Scatterplot and Boxplot



Stock Market data

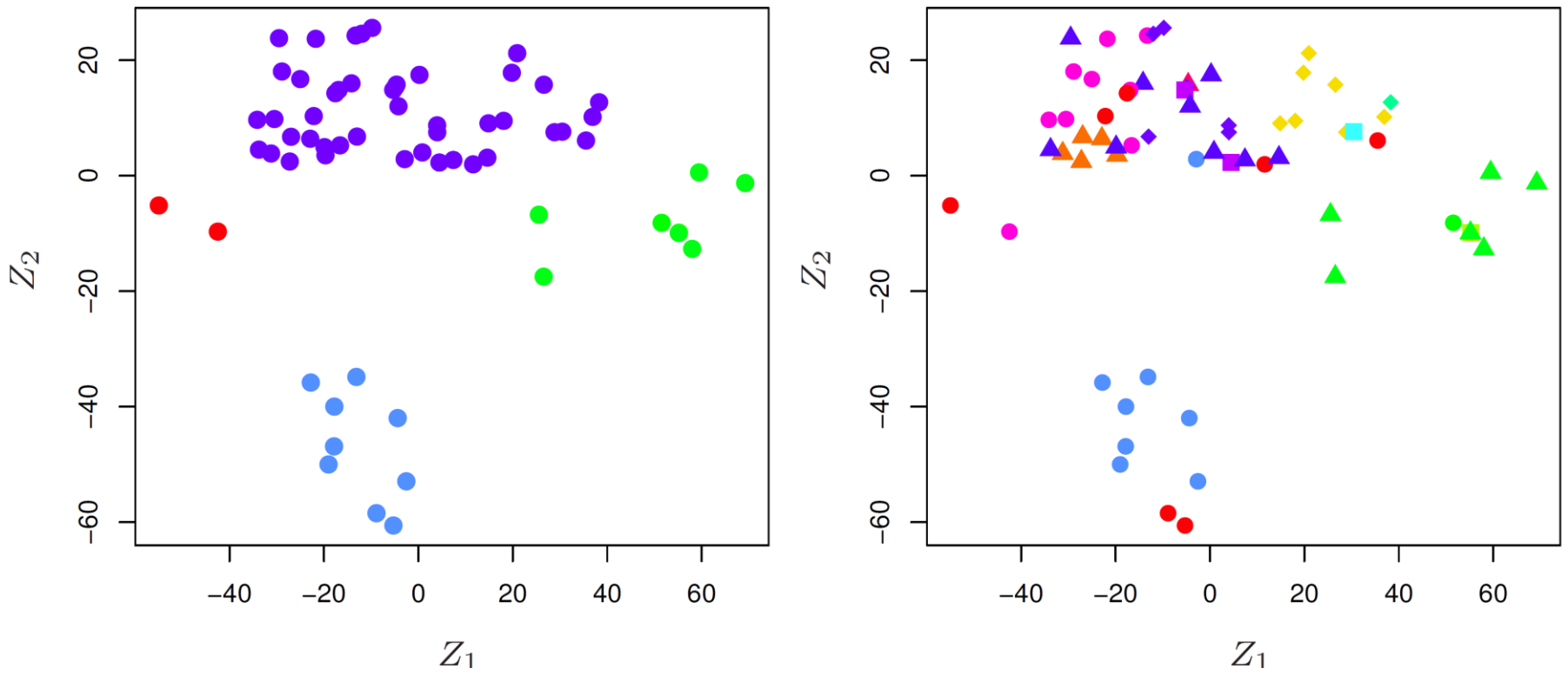
- Boxplot and heatmap





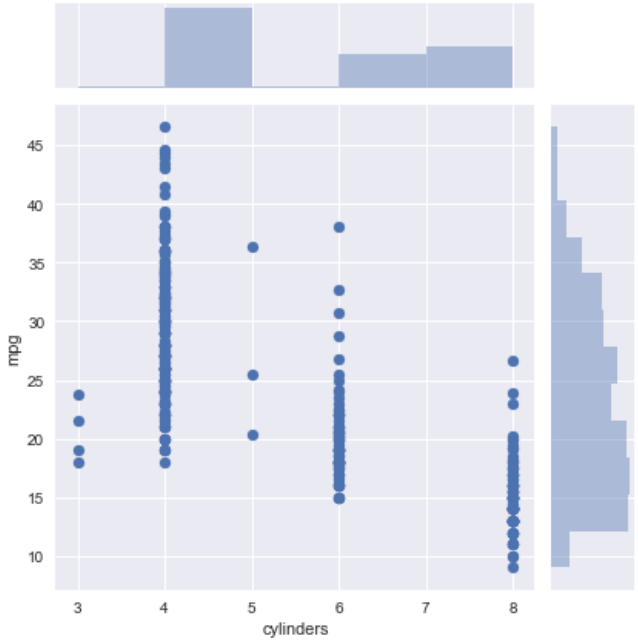
Gene Expression Data

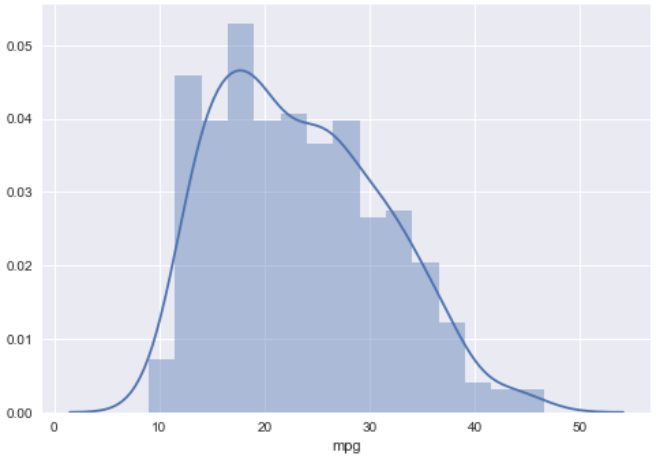
- Scatterplot

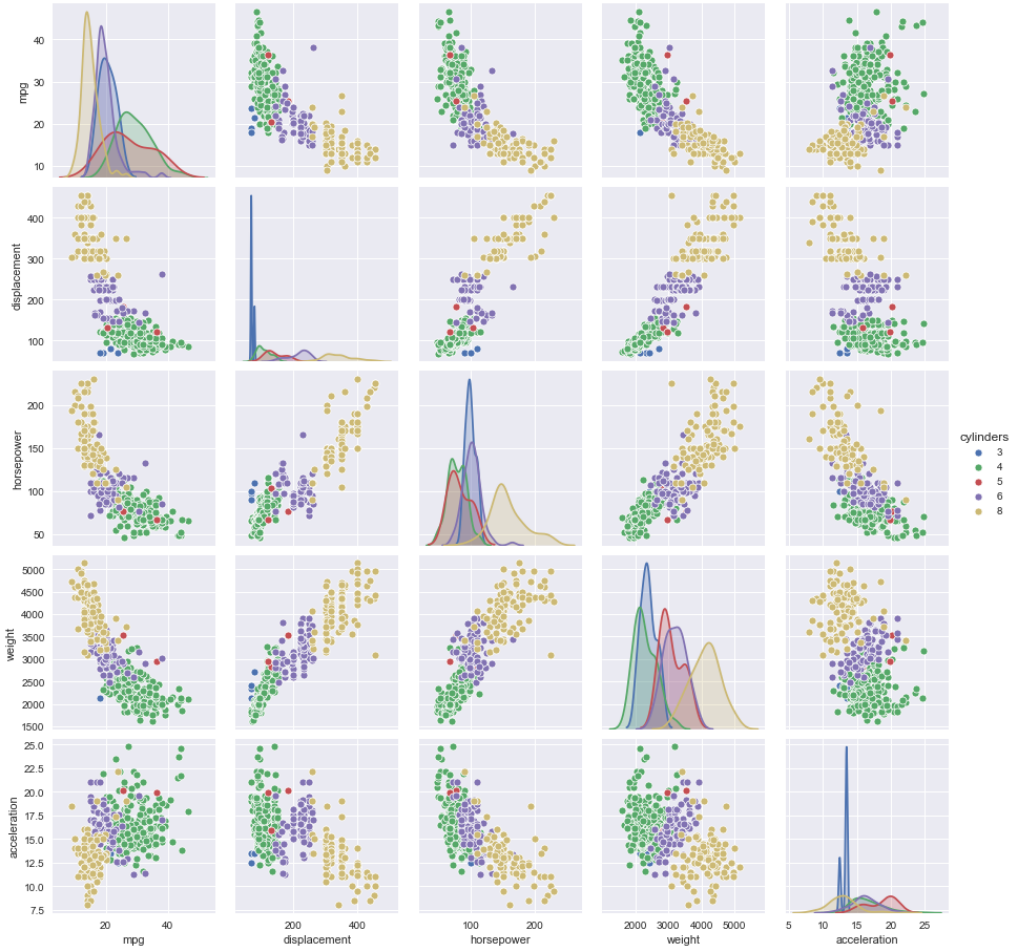


Auto data

- Pairplot, joinplot and displot

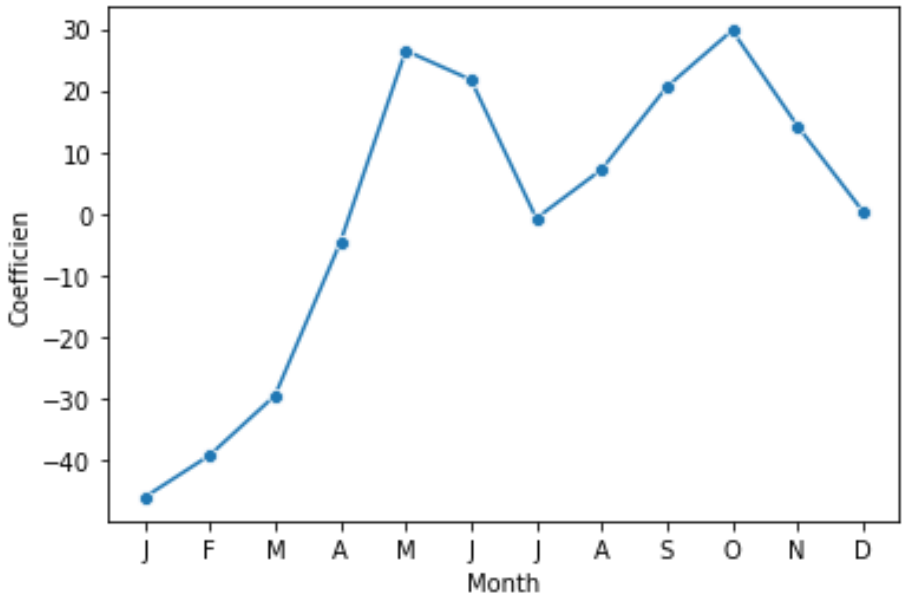


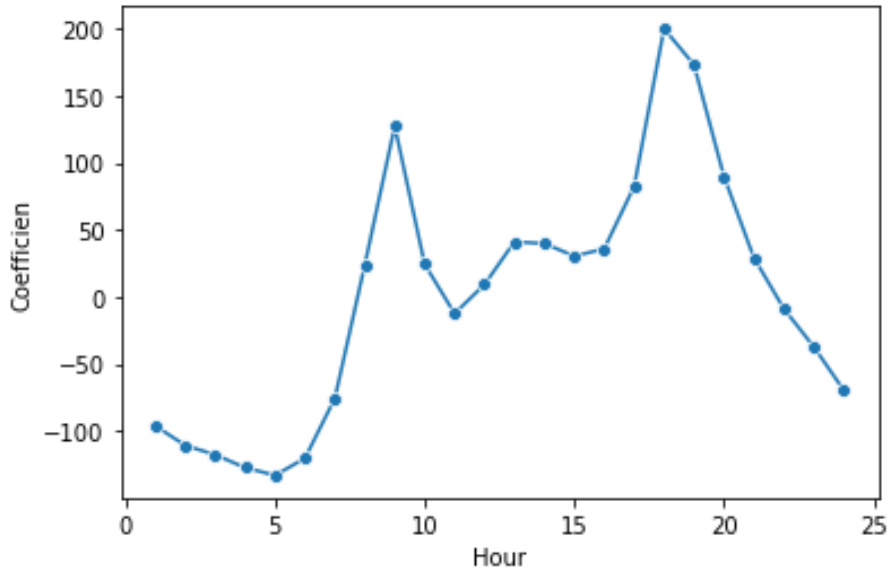




Bikeshare Data

- line plot





Multiple imputation

- It is common practice to perform multiple imputations, generating, for example, m separate imputations for a single feature matrix
  - Each of these m imputations is then put through the subsequent analysis pipeline (e.g. feature engineering, clustering, regression, classification). The m final analysis results (e.g. held-out validation errors) allow the data scientist to obtain understanding of how analytic results may differ as a consequence of the inherent uncertainty caused by the missing values. The above practice is called multiple imputation
- It is still an open problem as to how useful single vs. multiple imputation is in the context of prediction and classification when the user is not interested in measuring uncertainty due to missing values

Value normalization in deep learning

- In general, it isn't safe to feed into a neural network data that takes relatively large values or data that is heterogeneous (for example, data where one feature is in the range 0–1 and another is in the range 100–200). Doing so can trigger large gradient updates that will prevent the network from converging. To make learning easier for your network, your data should have the following characteristics:
  - Take small values—Typically, most values should be in the 0–1 range
  - Be homogenous—All features should take values in roughly the same range
- Additionally, the following stricter normalization practice is common and can help, although it isn't always necessary :
  - Normalize each feature independently to have a mean of 0
  - Normalize each feature independently to have a standard deviation of 1

Value normalization in deep learning

- In the MNIST classification, we started with image data encoded as integers in the 0–255 range, encoding grayscale values. Before we fed this data into our network, we had to cast it to float32 and divide by 255 so we'd end up with floating-point values in the 0–1 range
- Similarly, in regression problem, we started with features that took a variety of ranges—some features had small floating point values, and others had fairly large integer values. Before we fed this data into our network, we had to normalize each feature independently so that it had a standard deviation of 1 and a mean of 0

Confidence learning

- Confident learning (CL) has emerged as a subfield within supervised learning and weak-supervision to
  - Characterize label noise
  - Find label errors
  - Learn with noisy labels
- It can directly estimates the joint distribution of noisy and true labels and and ranking examples to train with confidence