CS 559: Machine Learning Fundamentals and Applications

Lecture 1: Overview of Machine Learning Project and Preprocessing

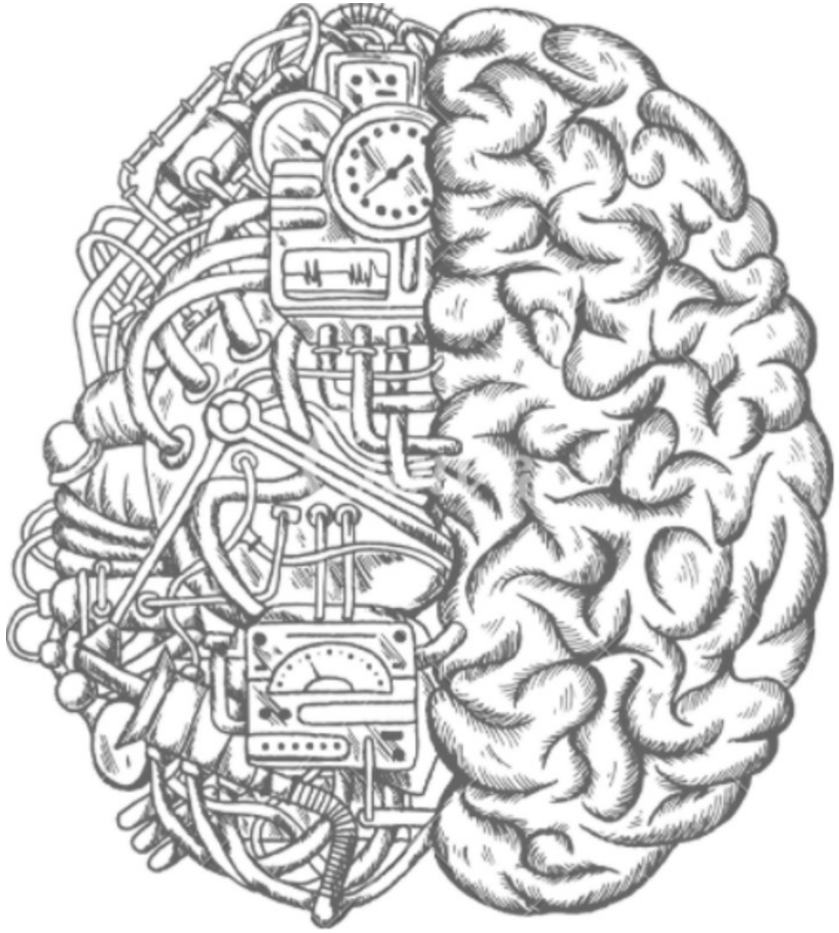
Outline

- Overview of Machine Learning Projects
- Preprocessing

Overview of Machine Learning Project

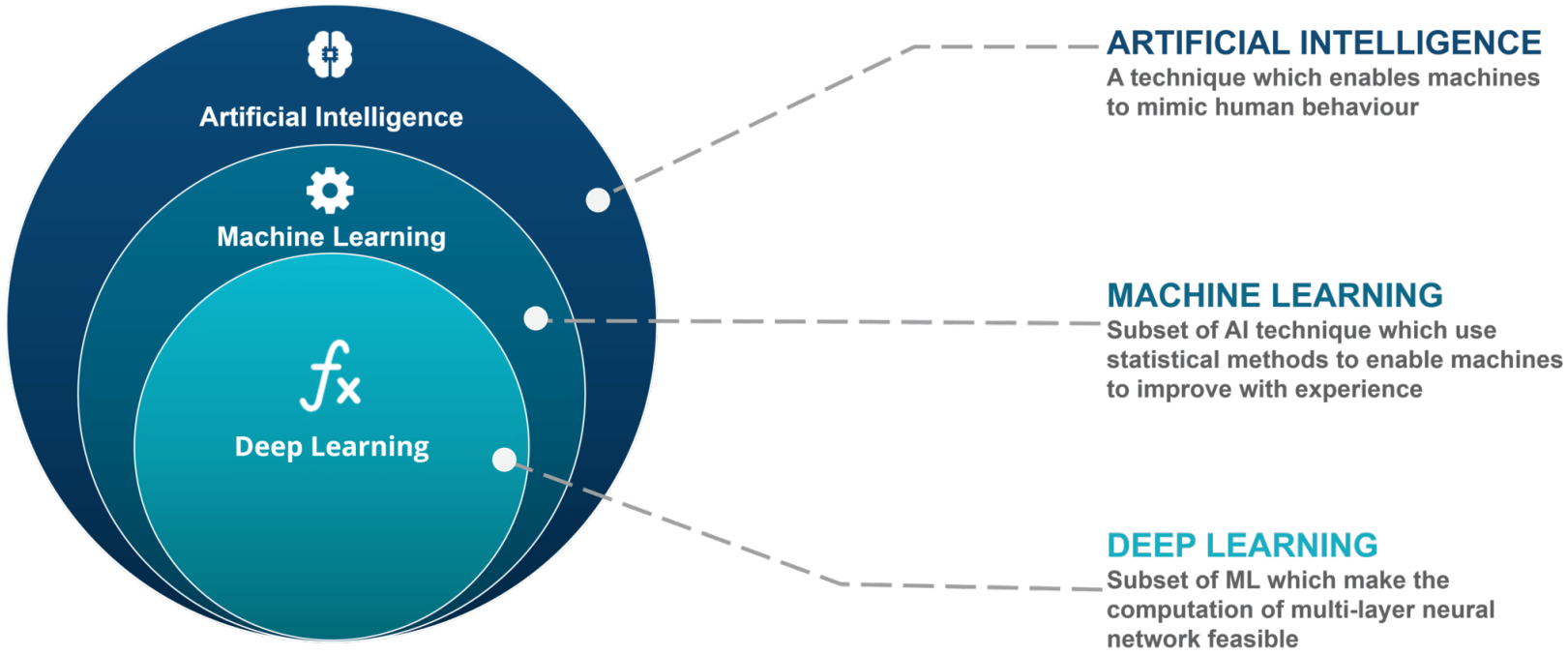
Machine Learning

- ML is everywhere!
  - Computer Science
  - Healthcare
  - Retail
  - Manufacturing
  - Energy
  - Financial Service
  - …



Machine Learning

- Why ML?



What is Machine Learning?

A computer program is said to learn from ***experience,*** *E,* with respect to some class of ***tasks,*** *T,* and performance ***measure,*** *P,* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

What is Machine Learning?

Machine Learning:

- The term first coined in 1959, by Arthur Samuel from IBM
- A branch of Artificial Intelligence (AI),
- Focused on design and development of algorithm
- Input: empirical data, such as that from sensors or databases,
- Output: ***patterns*** or ***predictions*** thought to be features of the underlying mechanism that generated the data.

Learner (the algorithm):

- Takes advantage of ***data*** to capture ***characteristics of interest*** of their unknown underlying probability distribution.

One fundamental difficulty:

- **Generalization**: The set of all possible behaviors given all possible inputs ***is too large*** to be included in the set of observed examples (training data). Hence the learner must ***generalize*** from the given examples in order to produce a useful output in new cases.

ML from Other Aspects

The Artificial Intelligence (AI) View:

- Learning is central to **human** knowledge and intelligence, and likewise, it is also essential for building **intelligent machines**.
- Years of effort in AI has shown that trying to build intelligent computers by programming all the rules cannot be done; automatic learning is crucial.
- For example, we humans are not born with the ability to understand language. ***We learn it*** and it makes sense to try to have computers learn language instead of trying to program it all it.

ML from Other Aspects

The Software Engineering View:

- Machine learning allows us to program computers by example, which can be easier than writing code in the traditional way.

The Statistics View:

- Machine learning is the marriage of computer science and statistics: computational techniques are applied to statistical problems.
- Machine learning has been applied to a vast number of problems in many contexts, beyond the typical statistics problems.
- Machine learning is often designed with different considerations than statistics (e.g., speed is often more important than accuracy).

Examples of ML

- Spam Filtering
- Goal: To decide whether it is spam from given emails knowing senders, titles, contents, etc.

Examples of ML

- Face Detection
- Goal: To identify people ID by recognizing faces

Examples of ML

- House Price Prediction
- Goal: To predict the house prices by conditions (e.g., room numbers, sizes, etc.)

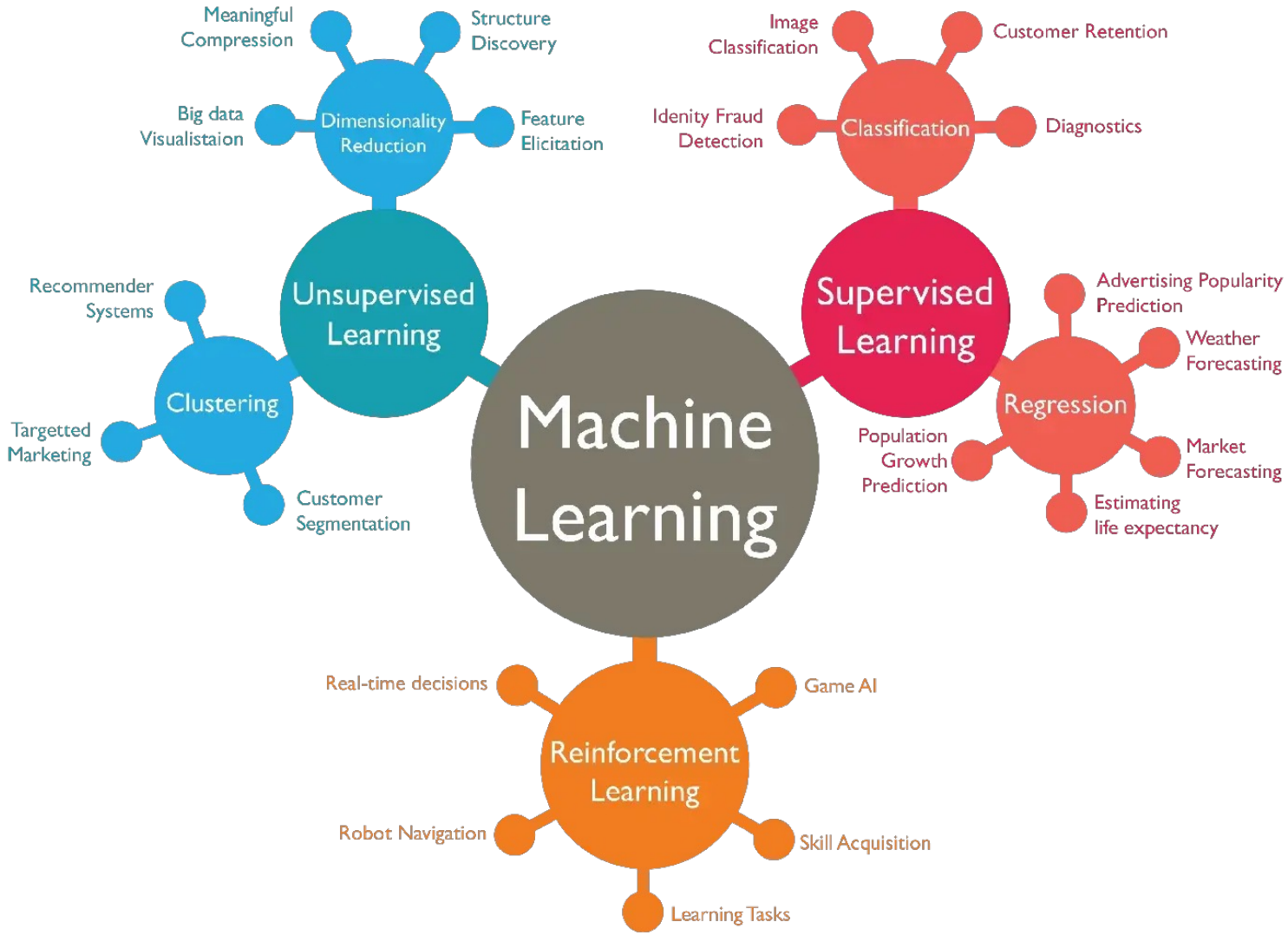
Examples of ML

- Games
- Goal: To calculate the probability of winning chance from the learner's move

Examples of ML

- Shopping
- Goal: To categorize fruits by characters (e.g., colors, size, etc.)

Learning Types of ML



Learning Types of ML

Supervised Learning

- Labeled Data
- Direct Feedback
- Predict outcome
- Forecast future

Unsupervised Learning

- No labels/targets
- No Feedback
- Find hidden structure in data

Reinforcement Learning

- Decision Process
- Reward system
- Learn series of actions

Typical Data Set for ML

Target

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BAY

Labels:

- headers
- column names
- feature names

Column: Features, predictors, attributes

Categorical – discrete data

- Integer (0 or 1)
- Text

Numerical – continuous data

Rows: observations, examples

Typical Data Set for ML

- Most of cases, common ML uses structured data, as seen in the previous slide.
  - Continuous
  - Categorical
  - Ordinal
  - Nominal
  - Unique
- Some data sets prohibit learnings from ML! We need to use deep learning instead.
  - Images
  - Videos
  - Sequential Data (e.g., Stock Price)
  - Texts

ML Project Workflow

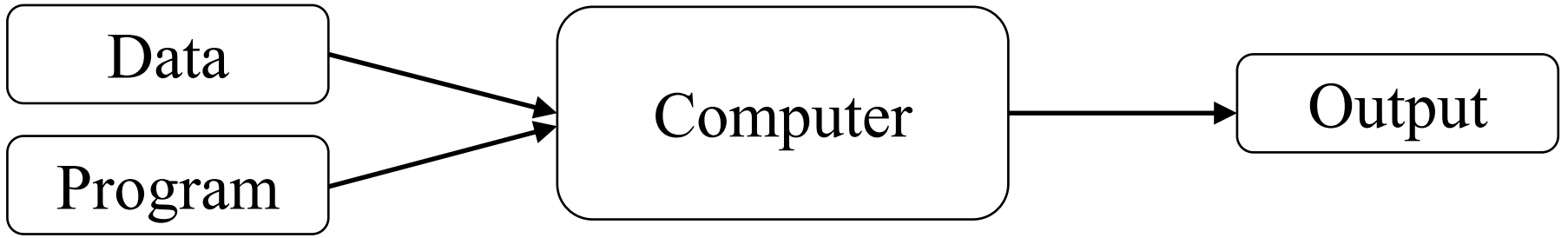
- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - How do we make models and what is after?
  - How can we make the models better?

ML Project Workflow

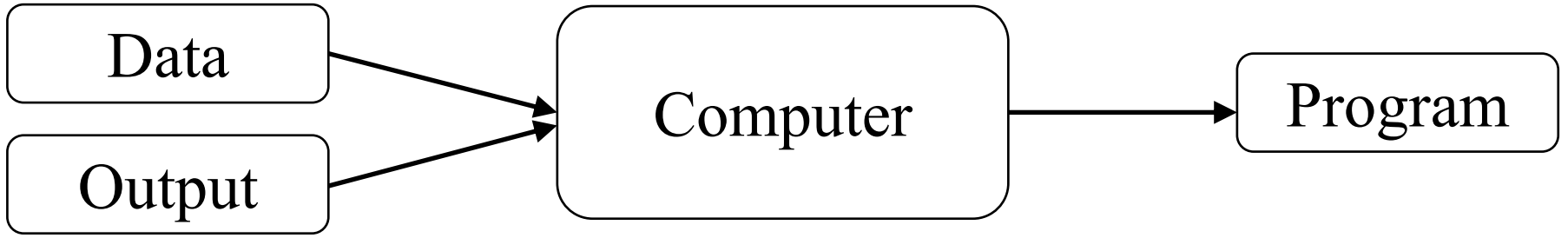
- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - How do we make models and what is after?
  - How can we make the models better?

ML vs Traditional Approach

- Traditional Programming



- Machine Learning



ML Project Workflow

- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - How do we make models and what is after?
  - How can we make the models better?

ML in Practice

ML is about:

- Given a collections of examples, called “training data”
- We want to predict something about novel examples, called “test data”

What we usually do:

- Build ***idealized models*** of the application area we are working in
- Develop algorithms and implement in code
- Use historical data to learn numeric parameters, and sometimes model structure
- Use test data to validate the learned model, quantitatively measure its predictions
- Assess errors and repeat…

ML in a Nutshell

- Every machine learning algorithm has three components:
  - Representation / Model Class
  - Evaluation / Objective Function
  - Optimization

Representation / Model Class

- Decision trees
- Sets of rules / Logic programs
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles

Evaluation / Objective Function

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence

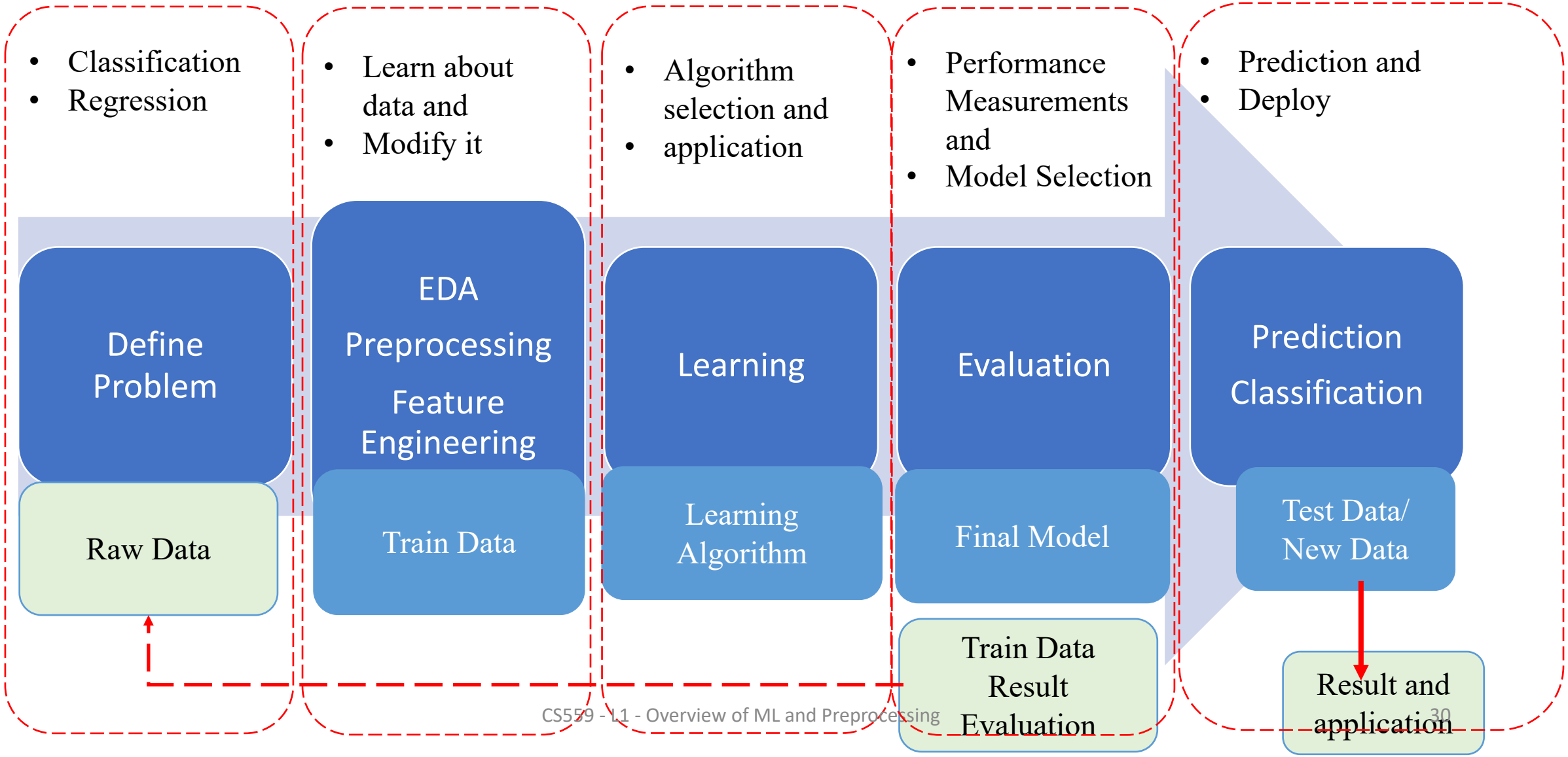
Optimization

- Discrete optimization
  - Minimal Spanning Tree
  - Shortest Path
- Continuous Optimization
  - Gradient Descent
  - Linear Programming

ML Project Workflow

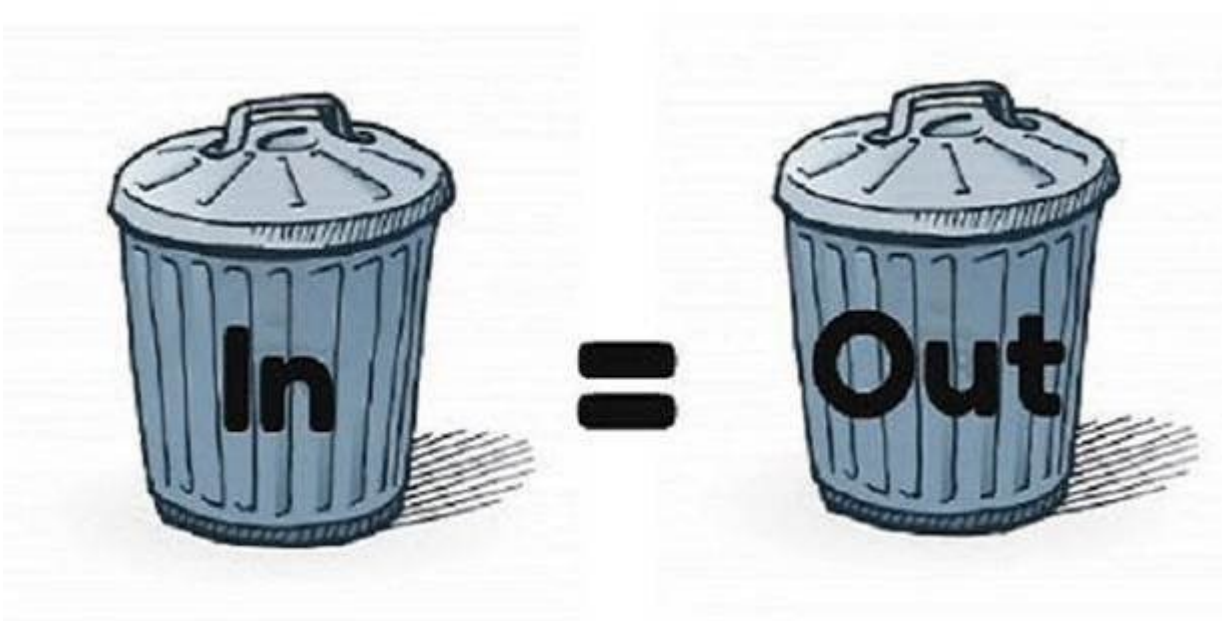
- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - How do we make models and what is after?
  - How can we make the models better?

Roadmap for ML



Importance of training data

- Data preprocessing is to make sure we have sensible data for ML!
- Exploratory Data Analysis (EDA) vs. Preprocessing vs. Feature Engineering
- EDA: learn about data
- Preprocessing: Data Cleaning and Transformation
- Feature Engineering: Tune features for algorithms



[https://www.linkedin.com/pulse/supply-chain-planning-garbage-out-cheryl-wiebe]

EDA Example

```
1 DF.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BAY

```
1 DF.shape
```

```
(20640, 10)
```

```
1 DF.dtypes
```

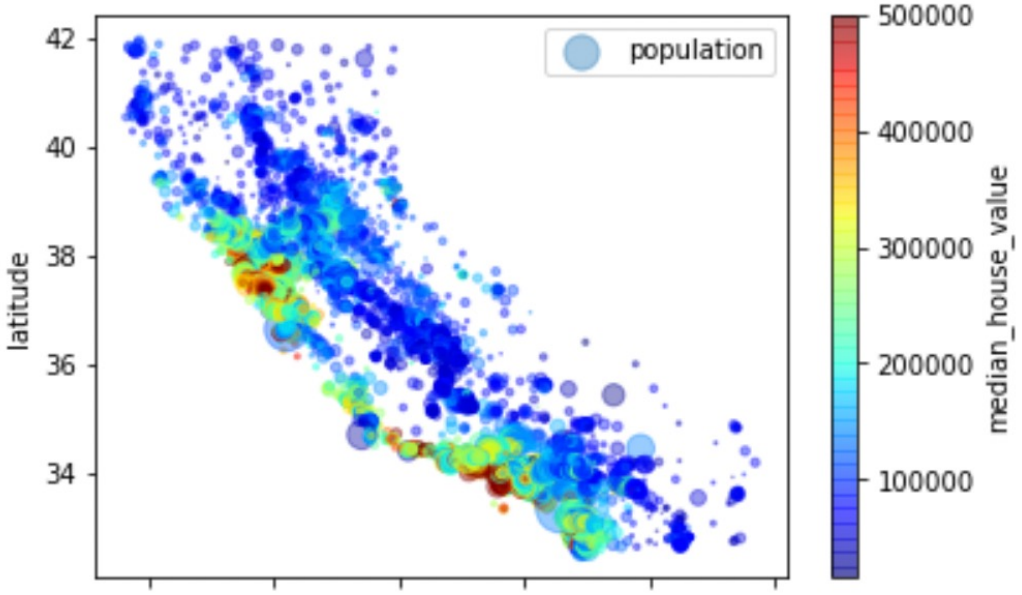
```
longitude             float64
latitude              float64
housing_median_age      int64
total_rooms             int64
total_bedrooms        float64
population              int64
households              int64
median_income         float64
median_house_value      int64
ocean_proximity        object
dtype: object
```

```
1 DF.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      207
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

EDA Example

```
DF.plot(kind="scatter",x="longitude",y="latitude", alpha=0.4, s=DF["population"]/100,
        label="population",c="median_house_value", cmap=plt.get_cmap("jet"),colorbar=True)
plt.legend()
plt.show()
```



Imputations

Missing values:

- The observation we intended to collect them but did not get them
  - Data entry issues, equipment errors, incorrect measurement, etc.
    - An individual may only have responded to specific questions in a survey, but not all
- Problems of missing data
  - Reduce the representativeness of the sample
  - Complicate data handling and analysis
  - Bias result from differences between missing and complete data

Imputations

1. Reducing the data set
   - Elimination of examples (rows) with missing values
   - Elimination of features (columns) with missing values
   - Either will reduce the sample size

2. Imputing missing values
   - Replace the missing value with the mean/median (continuous) or most common (categorical) value of that feature

3. Treating missing attribute values as a special value
   - Treat the missing value itself as a new value and be part of the data analysis
     - Make a simple model to estimate the missing value

Imputations

```
:  1  DF.isnull().sum()
```

```
:  longitude             0
   latitude              0
   housing_median_age    0
   total_rooms           0
   total_bedrooms      207
   population            0
   households            0
   median_income         0
   median_house_value    0
   ocean_proximity       0
   dtype: int64
```

Reducing the data set

```
In [6]:  1  DF_=DF.drop('total_bedrooms',axis=1)
         2  DF_.shape

Out[6]:  (20640, 9)
```

```
In [7]:  1  DF_=DF.dropna()
         2  DF_.shape

Out[7]:  (20433, 10)
```

```
In [9]:  1  DF_.shape[0]/DF.shape[0]

Out[9]:  0.9899709302325581
```

- Dropping the total bedrooms column is not a good idea.
- Dropping with the total bedrooms missing rows cannot hurt much.
  - In this case, it can be the easiest handling.

Imputations

```
1 DF['total_bedrooms'].describe()
```

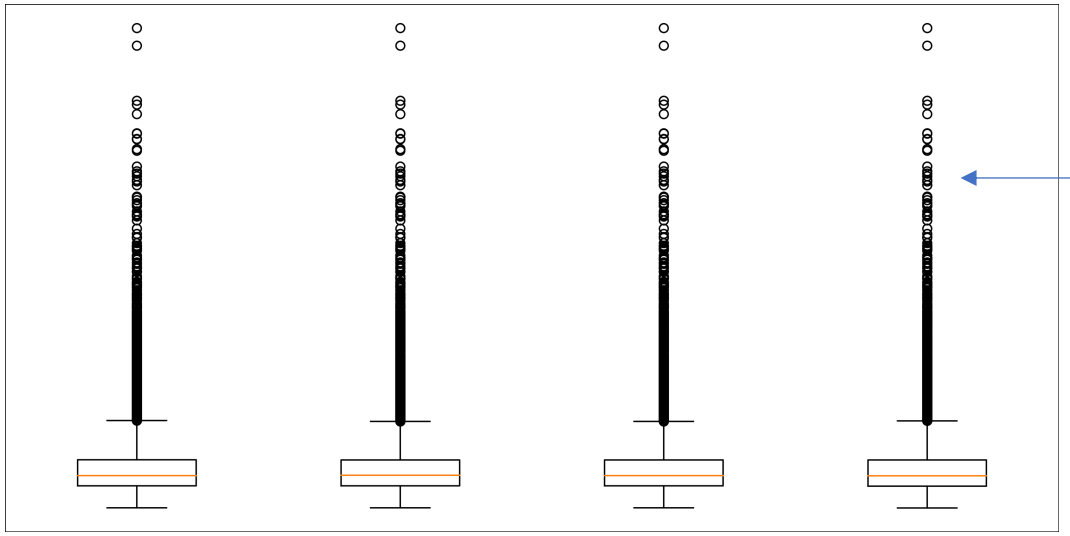
```
count    20433.000000
mean       537.870553
std        421.385070
min          1.000000
25%        296.000000
50%        435.000000
75%        647.000000
max       6445.000000
Name: total_bedrooms, dtype: float64
```

```
1 DF_ = DF
2 X0 = DF_['total_bedrooms'].dropna()
3 X1=DF_['total_bedrooms'].fillna(value=DF_['total_bedrooms'].mean())
4 X2=DF_['total_bedrooms'].fillna(value=DF_['total_bedrooms'].median())
5 X3=DF_['total_bedrooms'].fillna(value=0)
6 for i,x in zip(range(0,4),[X0,X1,X2,X3]):
7     print(f"The mean of X{str(i)} is {round(x.mean(),2)} and the median is {round(x.median(),2)}")
```

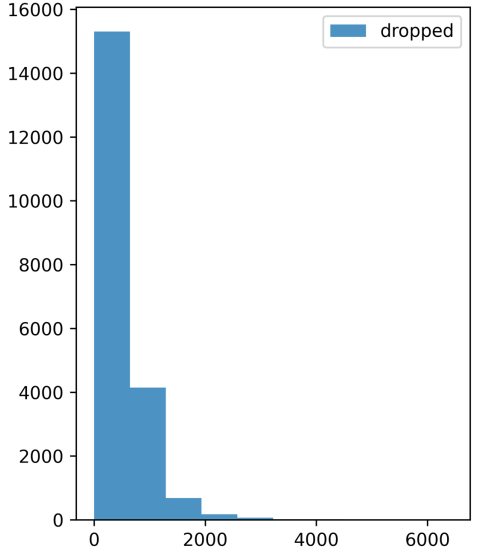
```
The mean of X0 is 537.87 and the median is 435.0
The mean of X1 is 537.87 and the median is 438.0
The mean of X2 is 536.84 and the median is 435.0
The mean of X3 is 532.48 and the median is 431.0
```

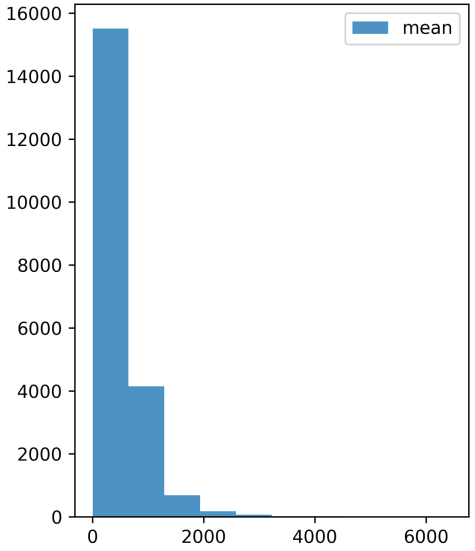
We do not see much differences in imputations!

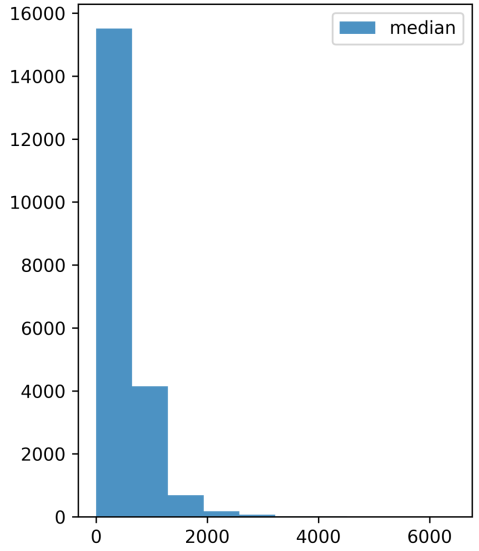
Imputations

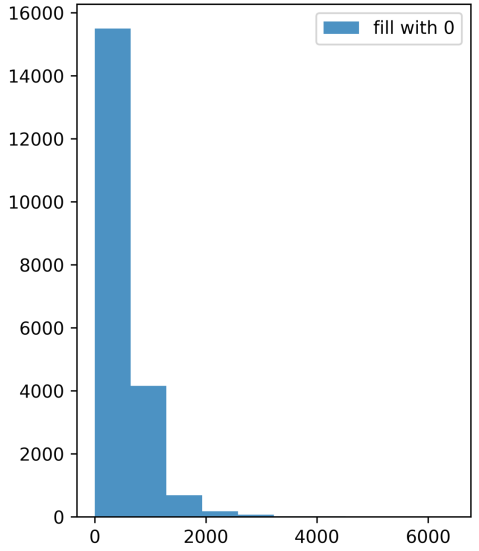


We can also remove outliers but usually based the target observations rather than feature-based.









Imputations

```
1 from scipy.stats import ks_2samp
```

```
1 for x in [X1,X2,X3]:
2     print(ks_2samp(X0,x))
```

```
KstestResult(statistic=0.0064533944747382 5, pvalue=0.7833422668265202)
KstestResult(statistic=0.0050113445076876 43, pvalue=0.9575858124995899)
KstestResult(statistic=0.0100290697674418 6, pvalue=0.2513034612542408 6)
```

- Kolmogorov-Smirnov (KS) test allows the 2-sample test as the greatest distance between the cumulative distribution function of each sample.
- Typically, we do not reject the null hypothesis that two distributions are the same if *p-value* is less than 5% (95% significance).

Data in different scale

Data in different scales

- Weight of a person (Pounds) vs weight of an elephant (US ton)
  - 1 US ton = 2000 Pounds
- For predicting weights for them, the error of elephant weights will significantly bias the prediction accuracy relative to the error for the persons weights

Data in different scale

- Approaches to bring different values onto the same scale
  - Normalization: rescale the feature to a range of [0,1]
  - Standardization: re-center the feature to the mean and scaled by variance

$$x_{norm}^{(j)} = \frac{x^{(j)} - x_{min}}{x_{\max} - x_{min}}$$

x_{min} and x_{max} are the min/max values of feature column $x^{(j)}$

$$x_{std}^{(j)} = \frac{x^{(j)} - \mu_x}{\sigma_x}$$

μ_x and σ_x are the mean and standard deviation of feature column $x^{(j)}$

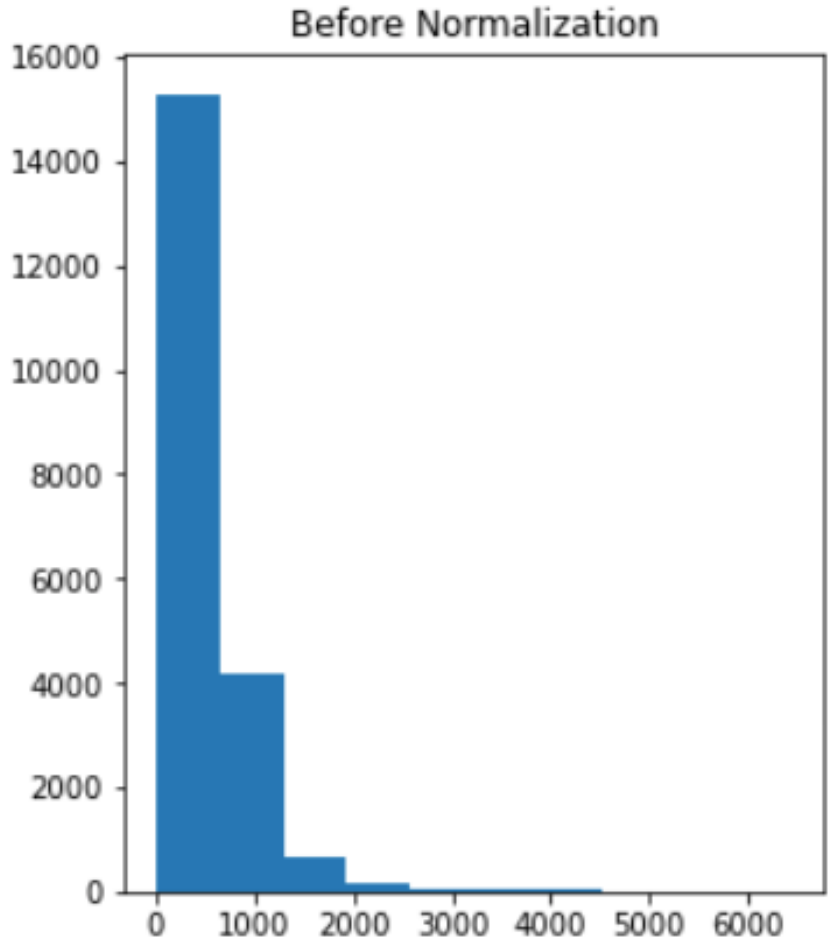
- Data scaling should be one of the first steps of data preprocessing for many machine learning algorithms
  - Some machine learning algorithms can handle data in different scales (e.g., decision trees and random forests)

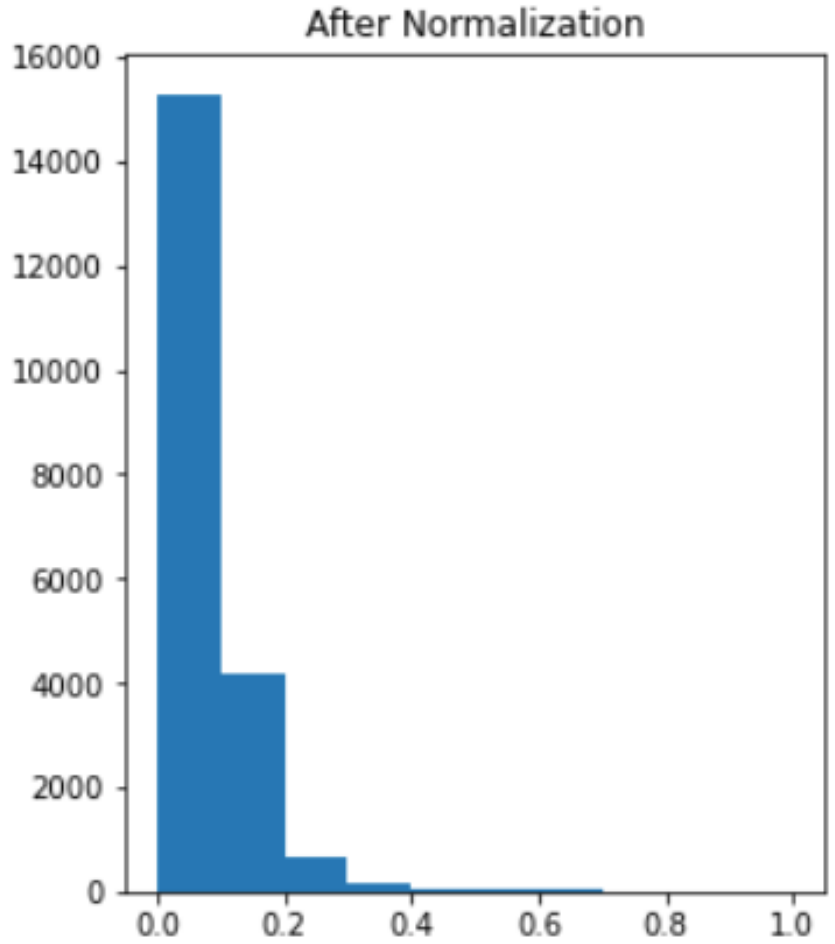
Data in different scale

- Standardization:
  - When measurements are in different units, we standardize the feature around the center 0 with 1σ.
  - Values at different scales can cause bias.
  - Assumes that data has a Gaussian distribution and if ML algorithm holds the assumption (e.g., Linear Regression, Logistic Regression, Linear Discriminant Analysis).

- Normalization:
  - To changes values to a common scale (between 0 and 1) without distorting differences in the ranges of values.
  - Typically used when features are in different ranges.
  - Use when distribution is not known or skewed.
  - K-Nearest Neighbors and Neural Networks

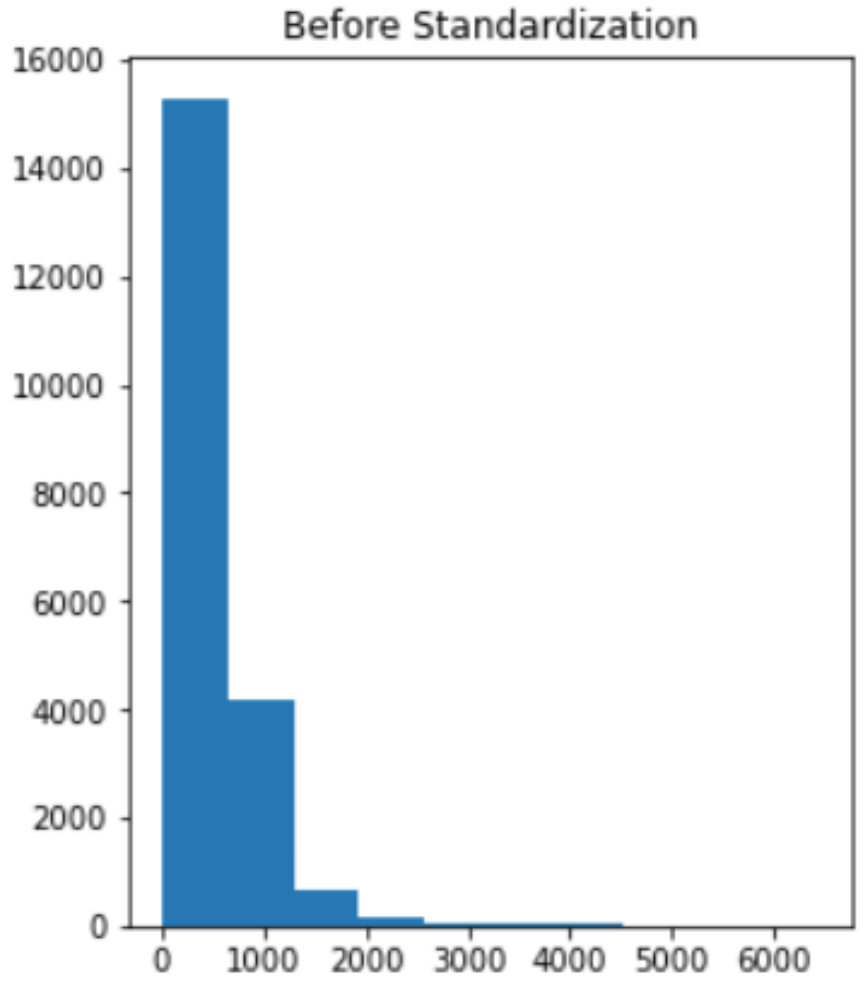
Data in different scale

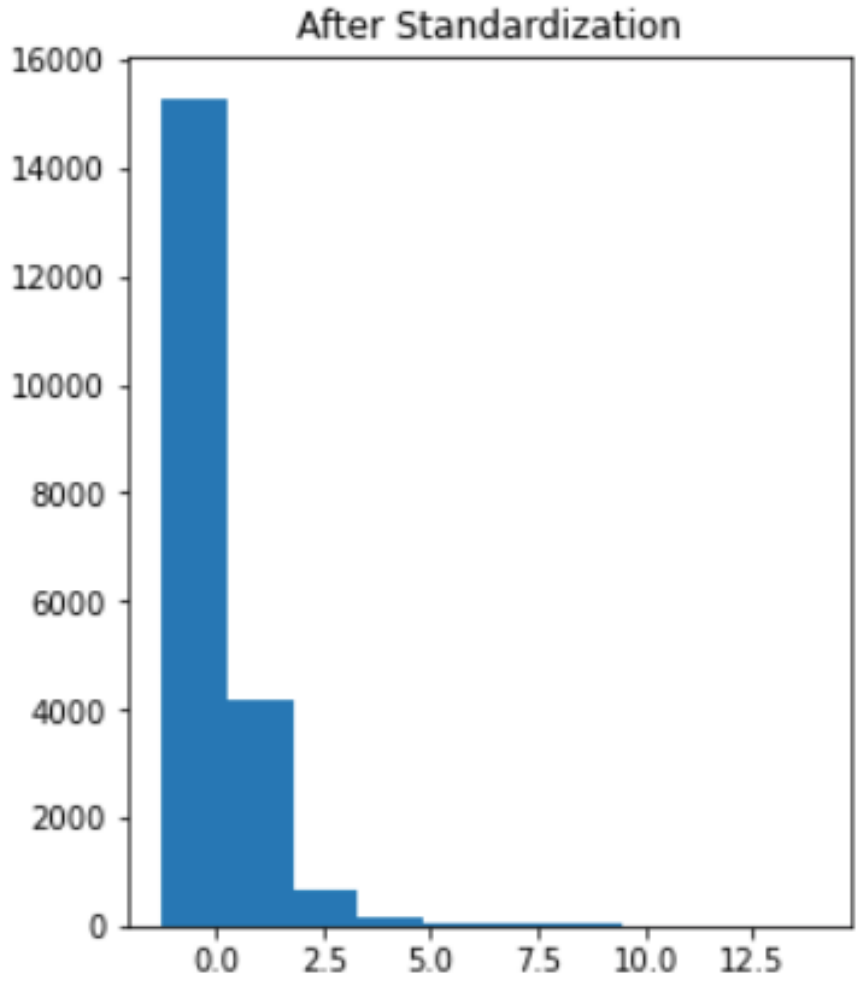





```
KstestResult(statistic=0.9899224806201551, pvalue=0.0)
```

Data in different scale



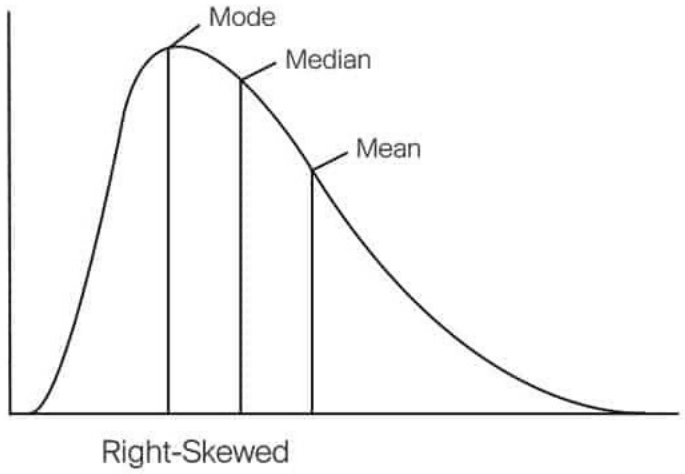


Handling Skewed Data

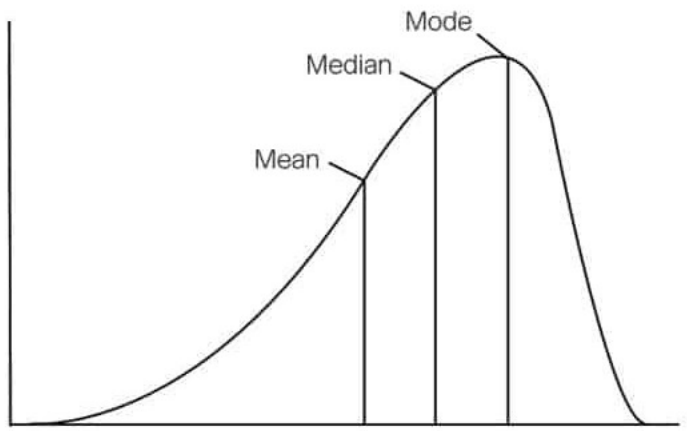
- Real-world data can be messy and contains attributes that need modifications before they can be used in modeling.
- In case of normal distribution, the mean, median, and mode are approximately close to each other at the center of distribution.
- The skewness of data can be determined by how these quantities are related to one another.

Handling Skewed Data

- Right skewed or Positive Skewed:
  - Mean > Median > Mode
- Left Skewed or Negative Skewed
  - Mode > Median > Mean
- The tail region may act outliers that can affect the model's performance in regression models.



Right-Skewed

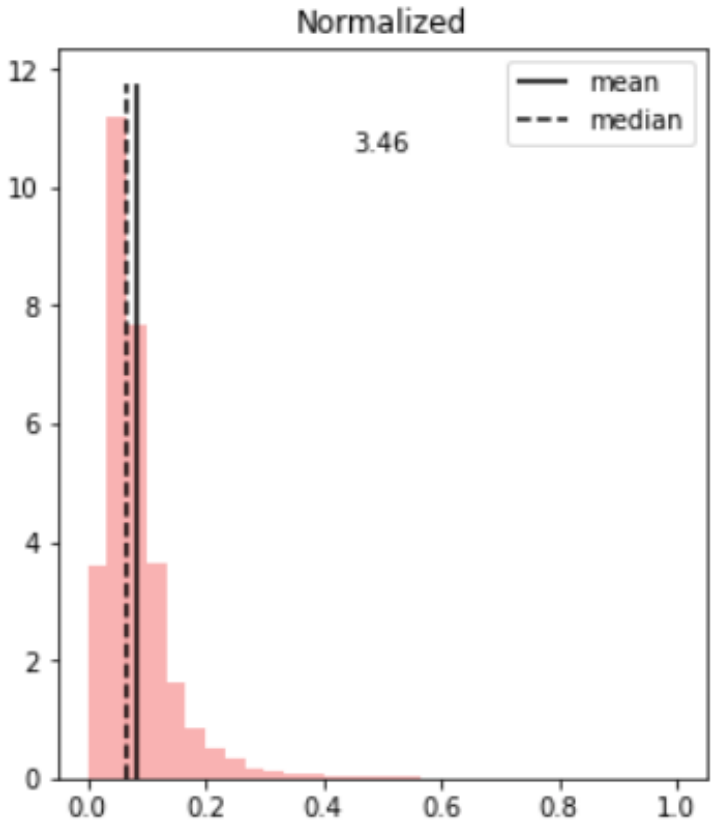


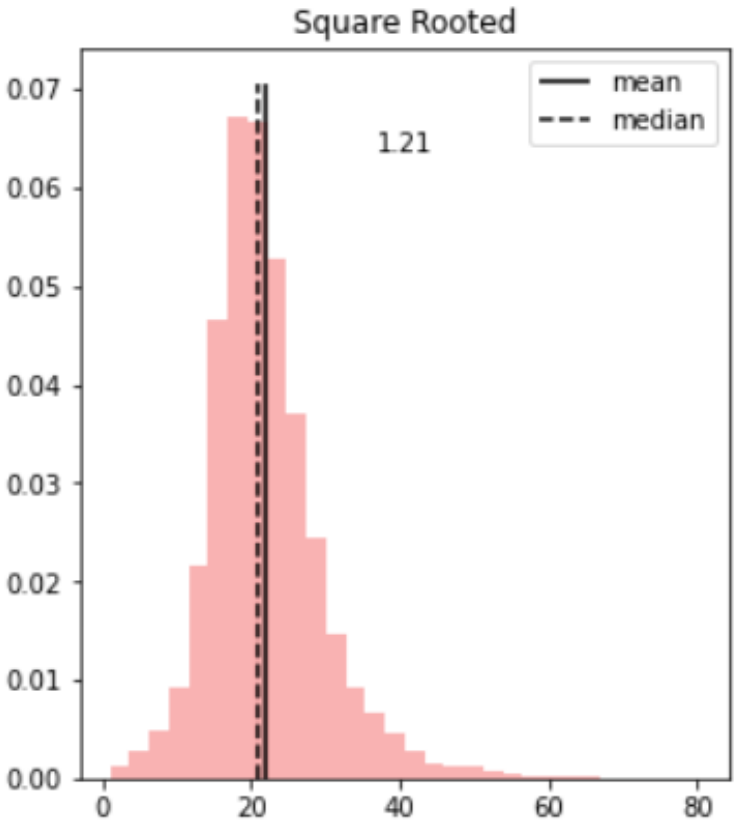
Left-Skewed (Negative Skewness)

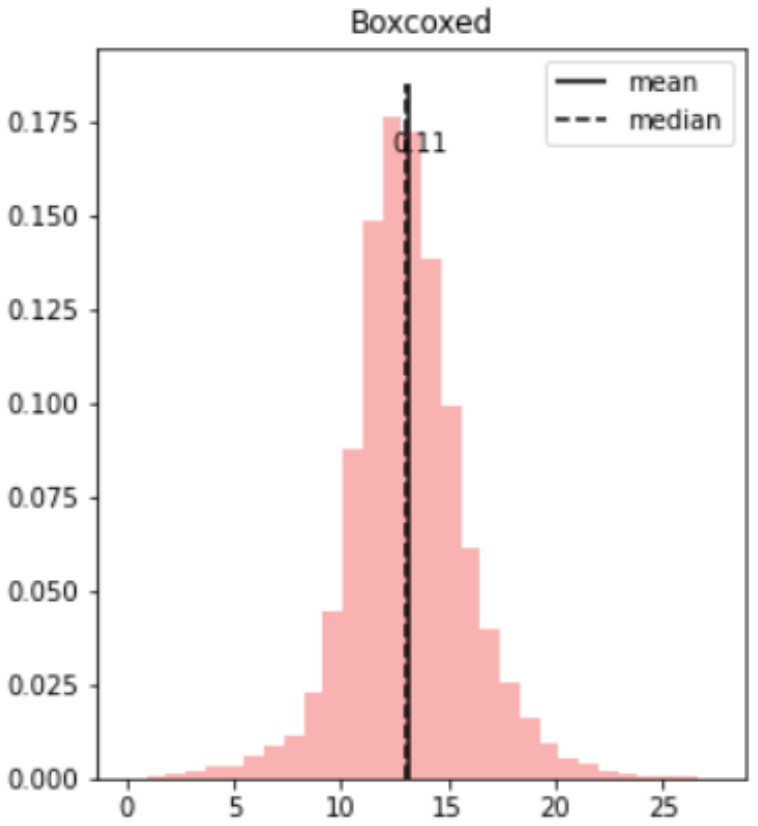
Handling Skewed Data

- Log transformation transforms skewed distribution to a normal distribution. (usually applies to right skewed data)
  - Values ≤ 0 cannot be transformed.
  - Add some constant so the minimum value be greater than 1 $\Rightarrow \log(1) = 0$
- Remove outliers (both)
- Normalize (applies to right skewed data)
- Cube root, square root (applies to right skewed data)
- Reciprocal (applies to right skewed data)
- Square (applies to left skewed data)
- Box Cox transformation (applies to both)
  - Transform using equations below:
  - $y(\lambda) = \begin{cases} (y^{\lambda} - 1)/\lambda & if\ \lambda \neq 0\ and\ y > 0 \\ \log y & if\ \lambda = 0\ and\ y > 0 \end{cases}$
  - $y(\lambda) = \begin{cases} ((y + \lambda_2)^{\lambda_1} - 1)/\lambda_1 & if\ \lambda_1 \neq 0\ and\ y < 0 \\ \log(y + \lambda_2) & if\ \lambda = 0\ and\ y < 0 \end{cases}$
  - Usually, $\lambda = [-5,5]$ but we use a λ value that gives the best approximation to a normal distribution.

Handling Skewed Data







Categorical data handling

- for ordinal data, convert the strings into comparable integer values
  - E.g., XL > L > M > S → 5 (XL) > 4 (L) > 3 (M) > 2 (S)
  - Note that the value of integer itself has no special meaning besides for ordering
  - Mapping needs to be unique: 1 to 1 mapping for going back and forth
- For nominal data, convert the strings into integers
  - E.g., Red (0), Blue (1), Green (2)
  - A common practice to avoid software glitches in handling strings
  - Note that the value of integer itself has no special meaning (non-comparable)
  - Mapping needs to be unique: 1 to 1 mapping for going back and forth
- To avoid mistakenly comparing encoded integers for nominal data, one-hot encoding can be used
  - Each unique value becomes a separate dummy feature

Categorical data handling


```
1 DF['ocean_proximity'].value_counts()
```

```
<1H OCEAN      9136
INLAND         6551
NEAR OCEAN     2658
NEAR BAY       2290
ISLAND            5
Name: ocean_proximity, dtype: int64
```

```
1 encoder = LabelEncoder()
2 housing_cat = DF["ocean_proximity"]
3 housing_cat_encoded = encoder.fit_transform(housing_cat)
4 housing_cat_encoded
```

```
array([3, 3, 3, ..., 1, 1, 1])
```

```
1 print(encoder.classes_)
```

```
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

```
1 set(housing_cat_encoded)
```

```
{0, 1, 2, 3, 4}
```

Categorical data handling

```
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = housing_cat_encoded.reshape(len(housing_cat_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
```

```
[[0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 ...
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
```

Correlation between features & Feature Engineering

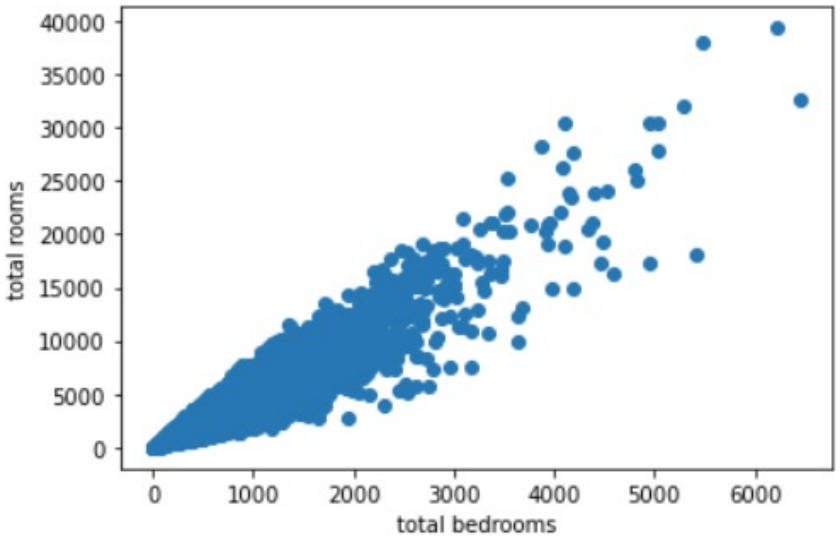
- One good way to reduce the data size
- Correlations between two features explains how they are related to each other.
  - Pearson correlation coefficient, $\rho_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y}$, is widely used.
  - Ranges from -1 to 1.
- Feature engineering extract features using domain knowledge
  - Improves the performance of ML
  - Sometimes can be considered as applied ML
- For example, if X and Y are tightly correlated
  - We can use only X as an independent variable
  - Or make a new feature call Z = XY as an independent variable

Correlation between features & Feature Engineering

```
1 corr_matrix = DF.corr()
2 corr_matrix[['total_bedrooms','total_rooms','households']]
```

	total_bedrooms	total_rooms	households
longitude	0.069608	0.044568	0.055310
latitude	-0.066983	-0.036100	-0.071035
housing_median_age	-0.320451	-0.361262	-0.302916
total_rooms	0.930380	1.000000	0.918484
total_bedrooms	1.000000	0.930380	0.979728
population	0.877747	0.857126	0.907222
households	0.979728	0.918484	1.000000
median_income	-0.007723	0.198050	0.013033
median_house_value	0.049686	0.134153	0.065843

Correlation between features & Feature Engineering




```
the correlation is 0.9303795046865079
```

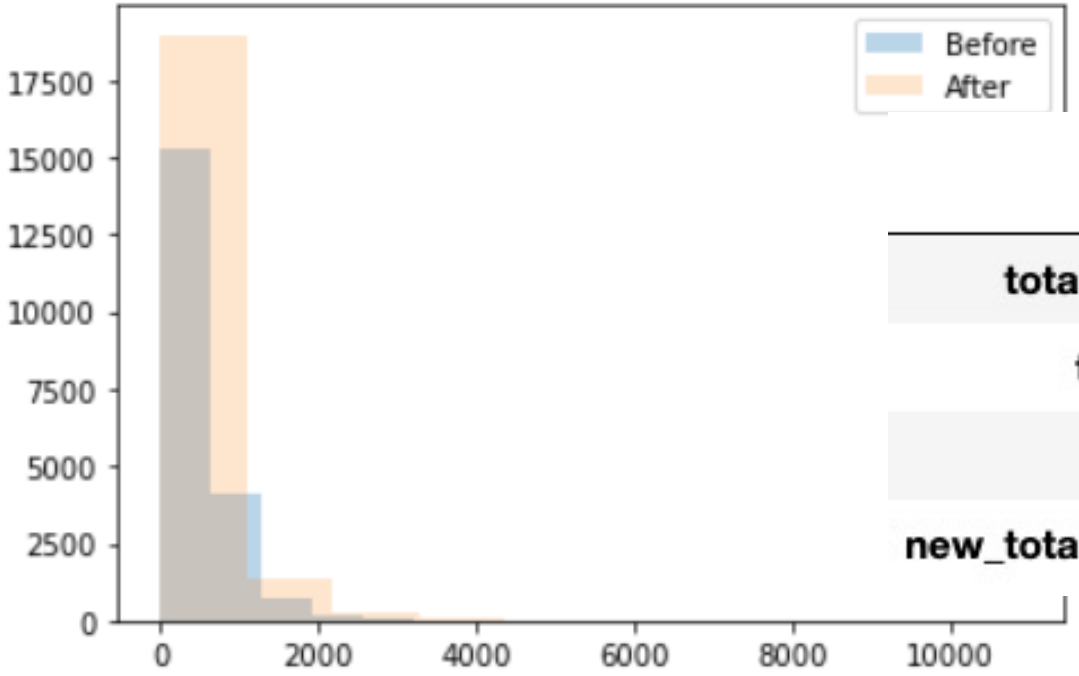
```python
df_x = DF[['total_bedrooms','total_rooms']]
df_x['new_total_bedrooms'] = df_x['total_rooms']*corr
df_x['total_bedrooms'] = df_x['total_bedrooms'].fillna(df_x['new_total_bedrooms'])
df_x.isnull().sum()
```

```
total_bedrooms        0
total_rooms           0
new_total_bedrooms    0
dtype: int64
```

Correlation between features & Feature Engineering

```
1 ks_2samp(DF['total_bedrooms'],df_x['total_bedrooms'])
```

```
KstestResult(statistic=0.01002906976744189, pvalue=0.2487036882495567)
```



	total_bedrooms	total_rooms	new_total_bedrooms
total_bedrooms	1.000000	0.930380	0.930380
total_rooms	0.930380	1.000000	1.000000
households	0.979728	0.918484	0.918484
new_total_bedrooms	0.930380	1.000000	1.000000

Correlation between features & Feature Engineering

```
X = df_x[['total_bedrooms','total_rooms','households']].dropna()
X['bedroom_per_room'] = X['total_bedrooms']/X['total_rooms']
X['bedrooms_per_household'] = X['total_bedrooms']/X['households']
X['total_rooms_per_household'] = X['total_rooms']/X['households']
X.corr()
```

	total_bedrooms	total_rooms	households	bedroom_per_room	bedrooms_per_household	total_rooms_per_household
total_bedrooms	1.000000	0.824827	0.870382	0.339474	0.273655	0.001655
total_rooms	0.824827	1.000000	0.918484	-0.120243	0.023986	0.133798
households	0.870382	0.918484	1.000000	0.042818	-0.044080	-0.080598
bedroom_per_room	0.339474	-0.120243	0.042818	1.000000	0.498175	-0.268805
bedrooms_per_household	0.273655	0.023986	-0.044080	0.498175	1.000000	0.657536
total_rooms_per_household	0.001655	0.133798	-0.080598	-0.268805	0.657536	1.000000

Preprocessing – Final Dataset

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
	-122.23	37.88	32.672294	8.208606	5.645033	7.675271	5.705034	2.728723
	-122.22	37.86	17.070476	11.413717	8.717972	11.487953	8.974699	2.724071
	-122.26	37.84	33.434943	9.799169	7.954136	10.099133	7.961036	0.797670
	-122.26	37.85	39.491212	8.560402	6.722807	9.055987	6.750997	0.822822

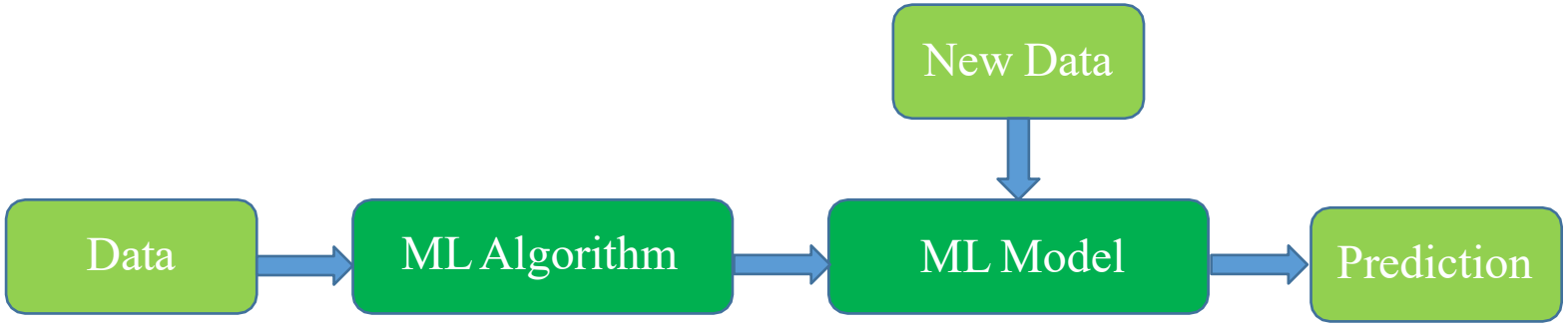
median_house_value	rooms_per_household	bedrooms_per_room	population_per_household	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
61.003713	3.113149	-2.957260	0.876915	0	0	0	1	0
58.036799	2.848110	-2.822167	0.707407	0	0	0	1	0
52.585885	2.062241	-1.811675	0.671329	0	0	0	1	0
47.360433	2.038908	-1.864239	0.905250	0	0	0	1	0

ML Project Workflow

- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - **How do we make models and what is after?**
  - How can we make the models better?

Machine learning, models and data

- Generalization - Machine learning is an algorithm that learns a model from data (training), so that the model can be used to predict certain properties about new data

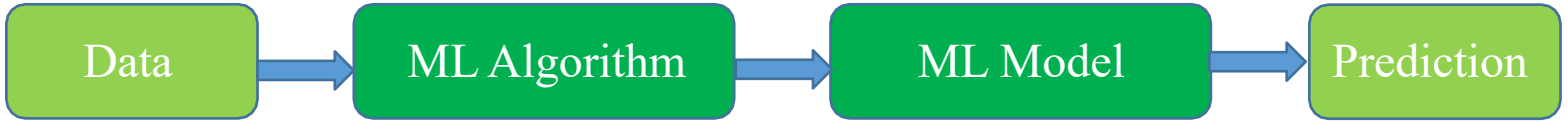


ML Project Workflow

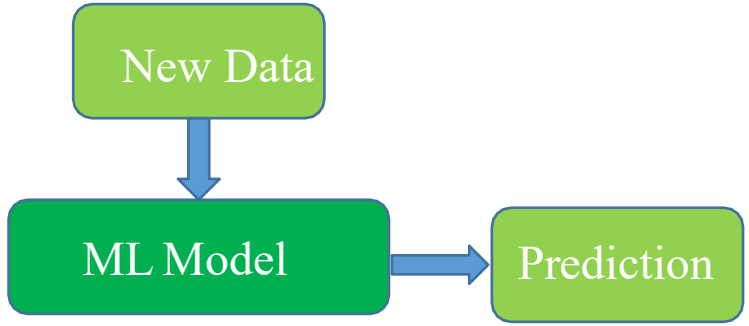
- What makes ML so special? Old School vs. New School
- What is the workflow in ML project?
  - What is preprocessing and exploratory data analysis (EDA)?
  - How do we make models and what is after?
  - **How can we make the models better?**

Training vs Inference

- Training is to build the ML model from data



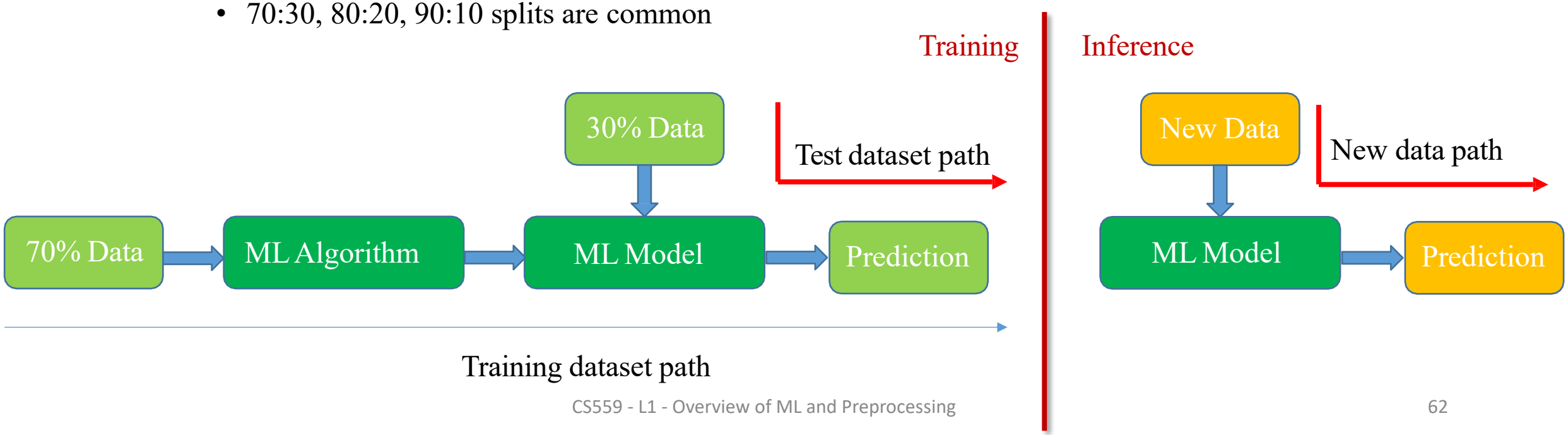
  - Typically, training is a one-time effort, but computationally intensive
  - Speed is a main concern
- Inference is to use the ML model to predict results for new data (generalization – most interesting for applications)



- Typically, inference is fast but happens more frequently with a lot of more new data (unlabeled)
- Scalability is a main concern

Split known data into training and test datasets

- Data known to ML model developers are split into two sets
  - Training dataset: data used to train the model
  - Test dataset: data used to give an indication on how well the trained model will generalize to new data (unknown at this point)
    - Test dataset is kept till the very end to evaluate the final model
    - Since test dataset withholds valuable information that the learning algorithm could benefit from, we don't want to put too much data into the test dataset either
      - 70:30, 80:20, 90:10 splits are common



Split known data into training and test datasets

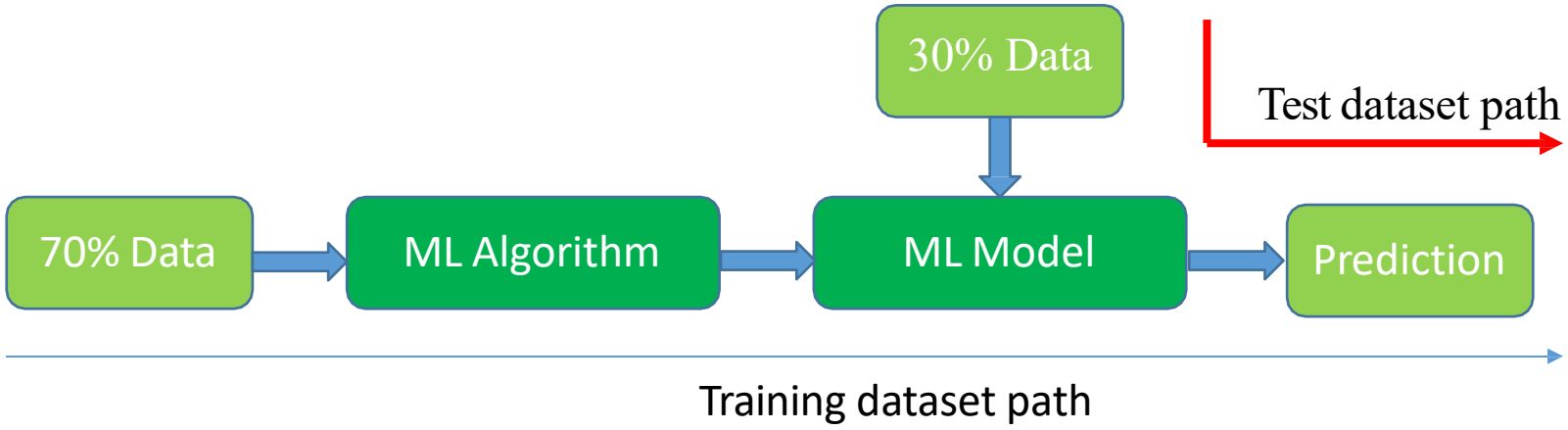

```
1 import numpy as np
2 from sklearn.model_selection import train_test_split
3 train_set, test_set = train_test_split(DF, test_size=0.3, random_state=42)
```

```
1 print(len(train_set),"train +", len(test_set),"test")
```

```
14448 train + 6192 test
```

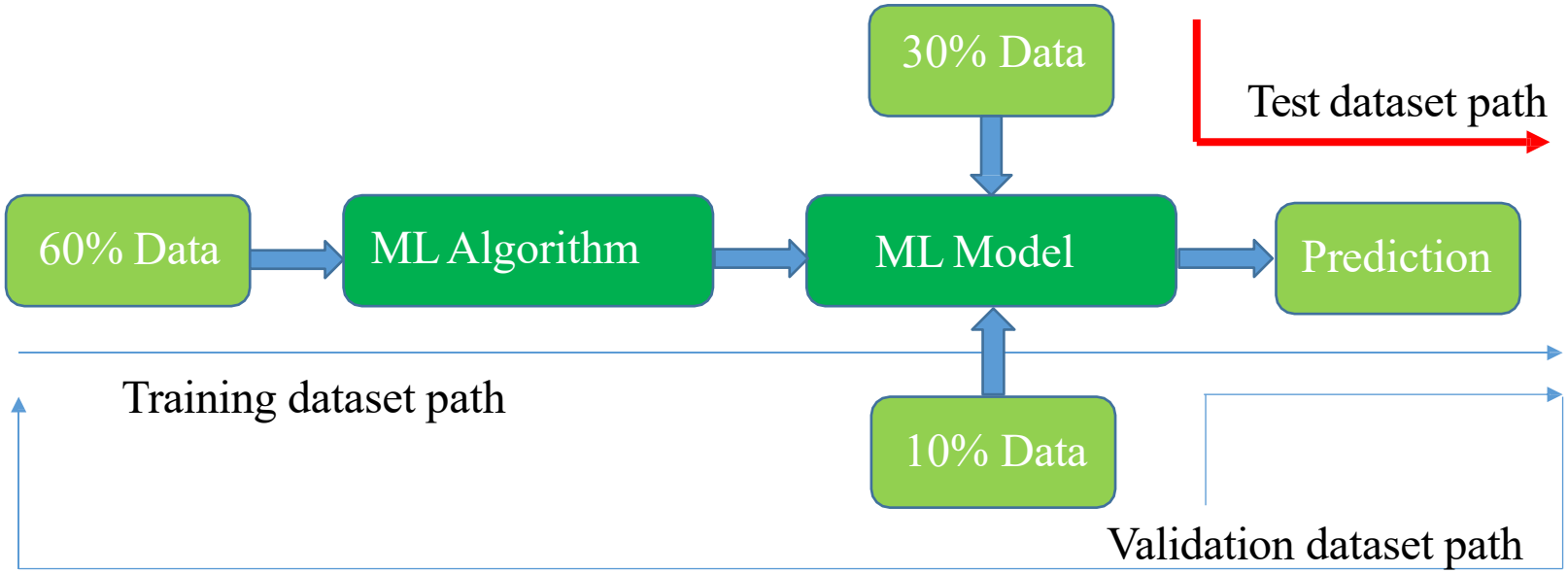
Cross-validation: a model tuning process

- How can we make the model training process to be aware of the targeted generalization quality so that training can do something about it?
- We need to put the predicted generalization results as part of the training optimization goal
  - We can NOT use the predicated generalization results from the test data, otherwise, the test data would become part of the training process
  - We want to keep the test data still independent of training so that its predication can still be a good indication of generalization quality for future unknown new data



Holdout cross-validation

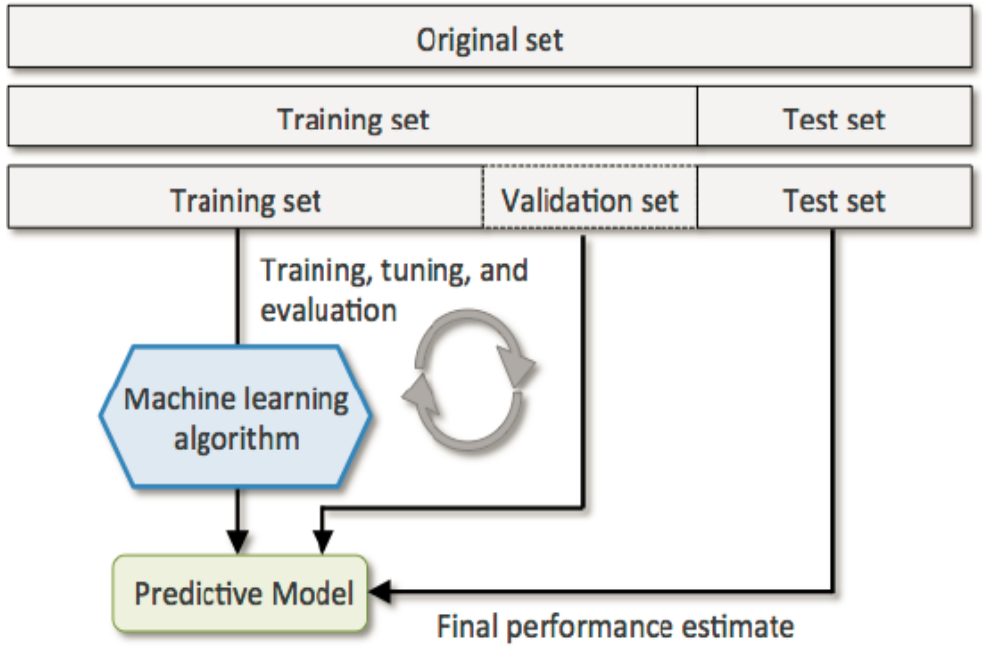
- Holdout cross-validation method
  - Training dataset is further split into two sets: training set + validation set



- Validation results are used to drive the continuation of training process
  - Until we obtain a reasonable validation result
- We still use test data to report the predicated generalization quality

Pros and Cons of holdout cross-validation

- Another view of the holdout cross-validation



- Pros: validation set is used to tune the model parameters for better generalization
- Cons: final results may be sensitive to how the dataset was split for validation

K-fold cross-validation

- Repeat holdout cross-validation k times on k subsets of the training data
  - Randomly split the training dataset into k folds without replacement
    - K-1 folds are used for training, and one fold used for validation
  - Repeat this k times so that we obtain k models
  - Typically k=10, but larger k for smaller dataset, and smaller k for larger dataset

- Pro: average performance from k models is less sensitive to the split
- Con: more computation time

Conclusion

ML Overview

- Machine Learning is everywhere!
- Garbage in Garbage out – ML does not over perform from the input.
- Pre-processing is important and the most time consuming part in ML.
- ML projects are broadly split into supervised learning and unsupervised learning.
- Splitting dataset to improve the performance is a standard way in ML.