Machine Learning and Deep Learning
Lecture-02

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Outline

- Intro to Machine Learning
- Data pre-processing
- EDA

Categories of ML

- Subcategories:
    - **Supervised** ML models are trained with labeled data sets.
    - **Unsupervised** ML models looks for patterns in unlabeled data.
    - **Reinforcement** ML trains machines through trial and error to take the best action by establishing a reward system.
        - Difficult to precisely specify the task.
        - Learning process of task can be dangerous.

Src: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

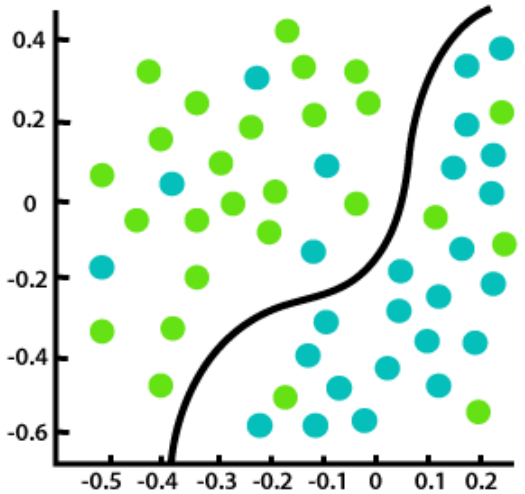
Supervised Learning

- Models are trained using "labeled" data, and predict output.
- Goal: Find a mapping function to map input var(x) with output var(y).
- E.g., training a model to recognize dog breeds from photos.
- Application: Image classification, Fraud Detection, spam filtering

Stanford Dogs dataset
(120 breeds)

Classification and Regression

- Process of finding a function which divides dataset into classes based on different parameters.
- Algorithms:
    - K-Nearest Neighbours
    - Support Vector Machines
    - Decision Tree Classification
    - Random Forest Classification



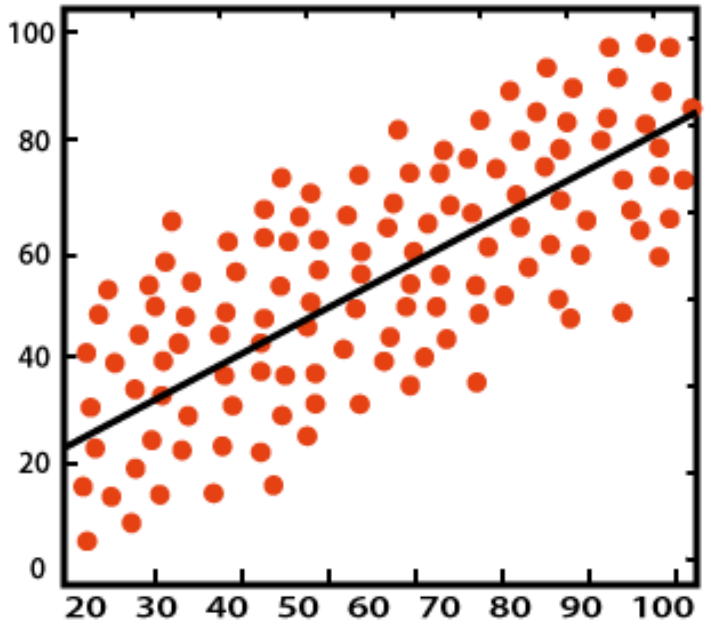
- Algorithm finds mapping function to map input(x) to discrete output(y).

- Application:
    - Image labeling, Email Spam Detection.

https://www.javatpoint.com/regression-vs-classification-in-machine-learning

Classification and Regression

- Process of finding correlations between dependent and independent variables.
- Predicts continuous variables.
- Algorithm:
    - Simple Linear Regression
    - Multiple Linear Regression
    - Support Vector Regression
- Finds mapping function to map input var(x) to continuous output var(y).

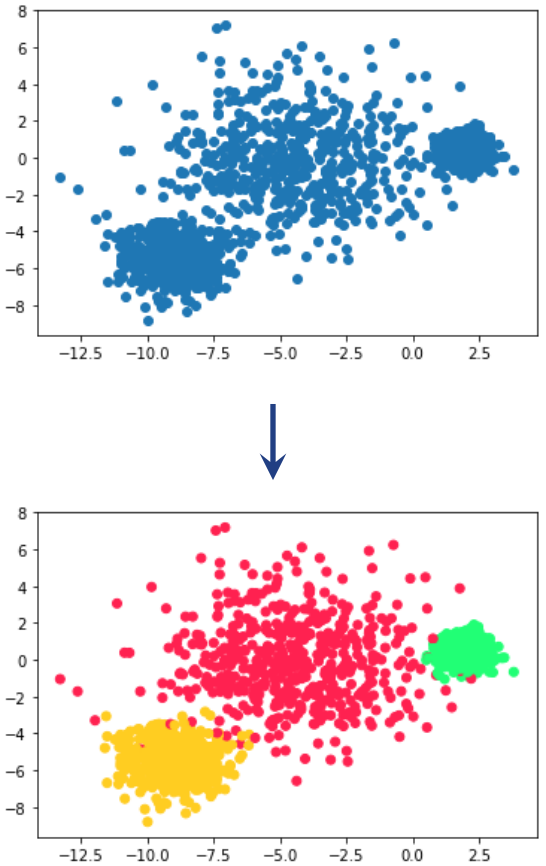


- Application:
    - Stock prices, temperature.

https://www.javatpoint.com/regression-vs-classification-in-machine-learning

Unsupervised Learning

- Labels are not provided, algorithm learns from data
- E.g., Finding an outlier in transactions.
- Algorithms:
  - Clustering: K-means, Hierarchical CA, DBSCAN
  - Dimensionality reduction: Principal components analysis (PCA)



Unsupervised Learning

- Labels are not provided
- Algorithm learns from data.

Anomaly detection

Src: Image from Géron, A. (2019)

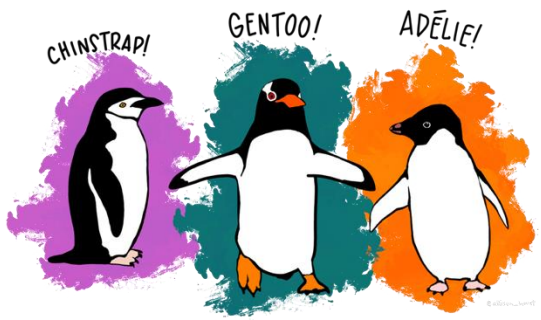
Data Pre-processing

Data Features

Labels

Features (also variables, attributes)

	species_short	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.100	18.700	181.000	3750.000	MALE
1	Adelie	Torgersen	39.500	17.400	186.000	3800.000	FEMALE
2	Adelie	Torgersen	40.300	18.000	195.000	3250.000	FEMALE
4	Adelie	Torgersen	36.700	19.300	193.000	3450.000	FEMALE
5	Adelie	Torgersen	39.300	20.600	190.000	3650.000	MALE



n: number of rows

m: number of features

https://github.com/allisonhorst/palmerpenguins

Data Types and Summary

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
 7   year               344 non-null    int64
dtypes: float64(4), int64(1), object(3)
memory usage: 21.6+ KB
```

df.describe()

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	year
count	342.000	342.000	342.000	342.000	344.000
mean	43.922	17.151	200.915	4201.754	2008.029
std	5.460	1.975	14.062	801.955	0.818
min	32.100	13.100	172.000	2700.000	2007.000
25%	39.225	15.600	190.000	3550.000	2007.000
50%	44.450	17.300	197.000	4050.000	2008.000
75%	48.500	18.700	213.000	4750.000	2009.000
max	59.600	21.500	231.000	6300.000	2009.000

df[feature].value_counts()

```
In [89]: df["species_short"].value_counts()

Out[89]: Adelie       146
         Gentoo       120
         Chinstrap     68
         Name: species_short, dtype: int64
```

Cleaning and Preprocessing Data

- Data requires some preprocessing and cleaning.
- Issues can be known in advance or noticed during exploration
- Common issues:
  - Missing values
  - Faulty data
  - Wrong data type
  - Duplicates

	species_short	island	culmen_length_mm	culmen_depth_mm
0	Adelie	Torgersen	39.100	18.700
1	Adelie	Torgersen	39.500	17.400
2	Adelie	Torgersen	40.300	18.000
3	Adelie	Torgersen	nan	nan
4	Adelie	Torgersen	36.700	19.300

Cleaning and Preprocessing Data

Some examples of ways to clean and preprocess data:

- Convert fields with text into numeric form
- Remove rows with missing / faulty data
- Fix rows with faulty datapoints
- Drop unnecessary variables (columns)
- Scaling or normalizing data
- Creating new variables
- Renaming variables (remove capitalization, spaces)

Removing Missing Values

	species_short	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.100	18.700	181.000	3750.000	MALE
1	Adelie	Torgersen	39.500	17.400	186.000	3800.000	FEMALE
2	Adelie	Torgersen	40.300	18.000	195.000	3250.000	FEMALE
3	Adelie	Torgersen	nan	nan	nan	nan	NaN
4	Adelie	Torgersen	36.700	19.300	193.000	3450.000	FEMALE

In this example we drop all rows with missing data df.dropna()

	species_short	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.100	18.700	181.000	3750.000	MALE
1	Adelie	Torgersen	39.500	17.400	186.000	3800.000	FEMALE
2	Adelie	Torgersen	40.300	18.000	195.000	3250.000	FEMALE
4	Adelie	Torgersen	36.700	19.300	193.000	3450.000	FEMALE
5	Adelie	Torgersen	39.300	20.600	190.000	3650.000	MALE

Imputing missing values

- Replace missing values by mean or median of the features.

	species_short	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	0	2	39.1	18.7	181.0	3750.0	2
1	0	2	39.5	17.4	186.0	3800.0	1
2	0	2	40.3	18.0	195.0	3250.0	1
3	0	2	NaN	NaN	NaN	NaN	3
4	0	2	36.7	19.3	193.0	3450.0	1
...	...	...	...	...	...	...	...
339	2	0	NaN	NaN	NaN	NaN	3

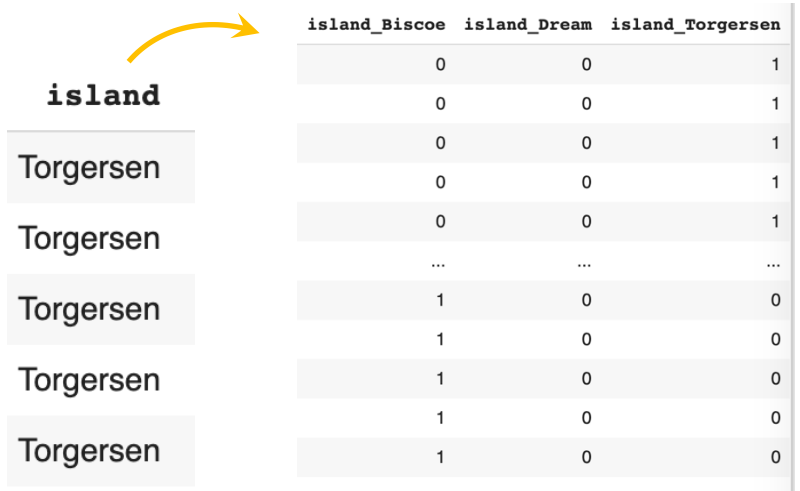
	species_short	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	0	2	39.1	18.7	181.0	3750.000000	2
1	0	2	39.5	17.4	186.0	3800.000000	1
2	0	2	40.3	18.0	195.0	3250.000000	1
3	0	2	NaN	NaN	NaN	4201.754386	3
4	0	2	36.7	19.3	193.0	3450.000000	1
...	...	...	...	...	...	...	...
339	2	0	NaN	NaN	NaN	4201.754386	3

```
df["body_mass_g"] = imp.fit_transform(df[["body_mass_g"]])
```

Changing categorical to numerical values

- Datasets can contain categorical and ordinal features with text.
- These need to be converted to numeric form

- Several ways are:
  - **One-hot encoding**
  - Label encoding
  - Ordinal encoding



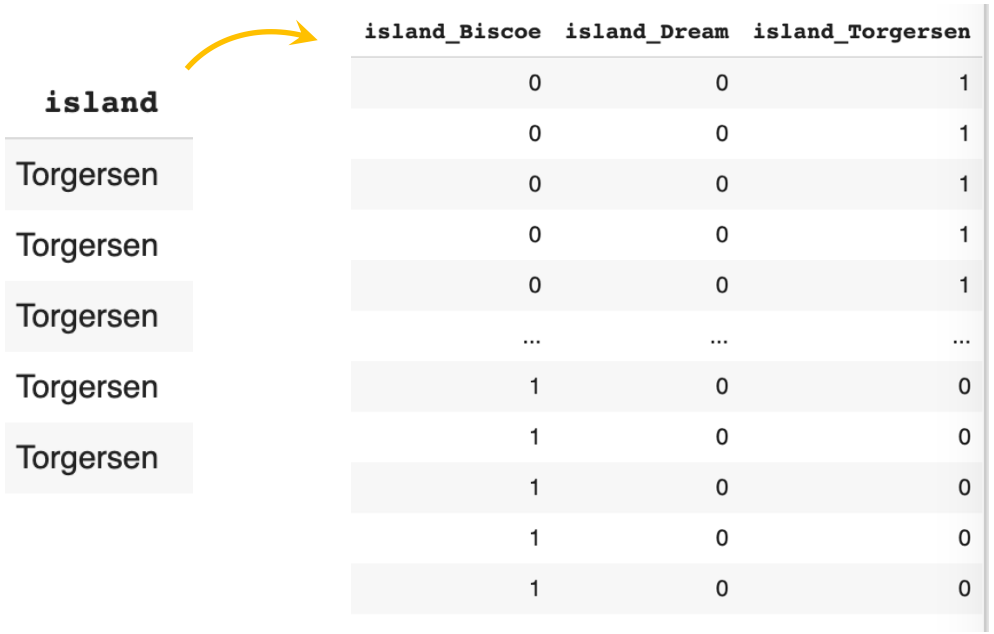
island
Torgersen
Torgersen
Torgersen
Torgersen
Torgersen

island_Biscoe	island_Dream	island_Torgersen
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
...	...	...
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0

Categorical Variables: Variables that take on names or labels.
Ex: Marital status (“married”, “single”, “divorced”)

Dummy variables and one-hot encoding

- A dummy variable is a binary variable (0/1)
- One-hot encoding transforms each possible value from a categorical variable into a new binary column.



island
Torgersen
Torgersen
Torgersen
Torgersen
Torgersen
Torgersen

island_Biscoe	island_Dream	island_Torgersen
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
...	...	...
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0

pd.get_dummies()

sklearn.preprocessing.OneHotEncoder()

Label encoding

- Instead of creating columns with binary encoding, each possible value is given a unique integer value
- Penguins case: Torgensen = 2, Biscoe = 1, Dream = 0
- Issue: Algorithms might misjudge there to be an order or relationship in the data that does not really exist!!

sklearn.preprocessing.LabelEncoder()

Ordinal encoding

- Very similar to label encoding, but specifically for maintaining order of the values.
- Since the penguin dataset has no text variable that is ordinal, example given demonstrate using body_mass_g

body_mass_g	size
3750.0	30
3800.0	32
3250.0	11
3450.0	18
3650.0	26
...	...

sklearn.preprocessing.OrdinalEncoder()

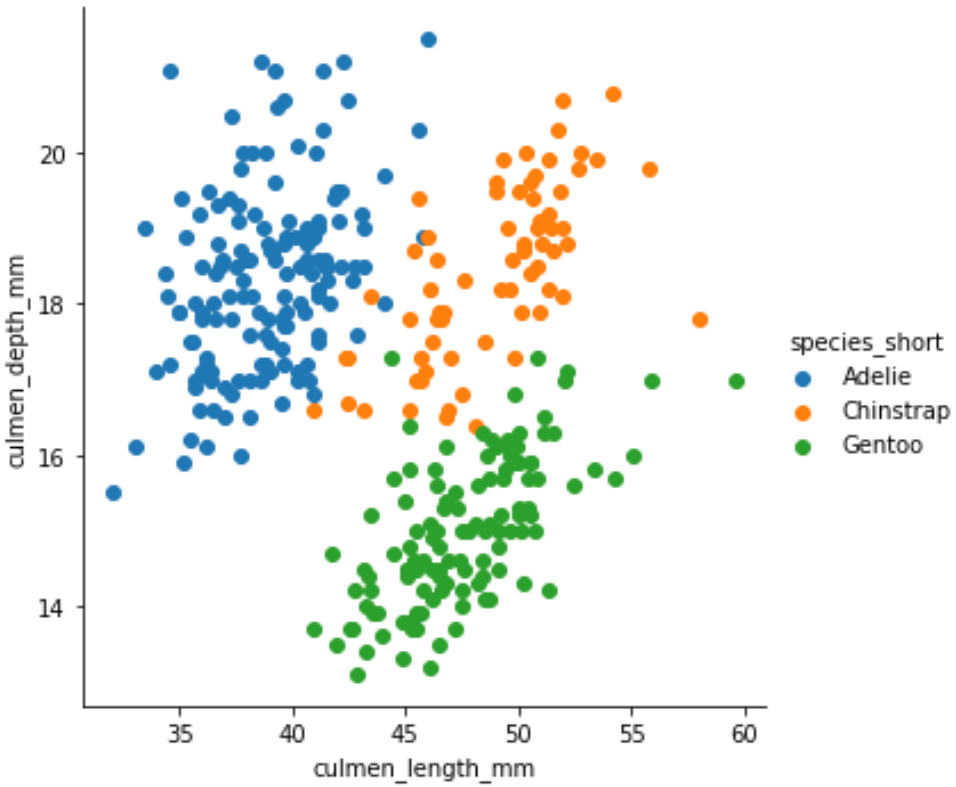
Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA)

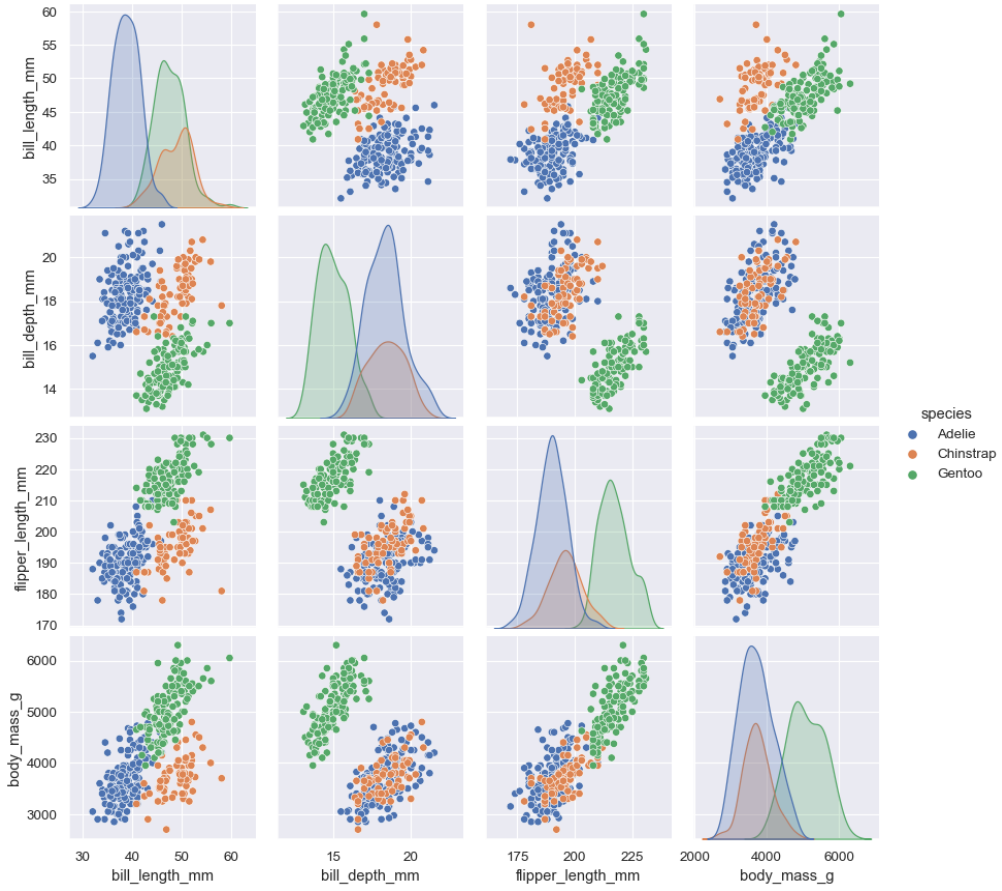
- An approach for data analysis that employs a variety of techniques (mostly graphical) to
  - uncover underlying structure;
  - extract important variables;
  - detect outliers and anomalies;
  - test underlying assumptions;
  - develop parsimonious models; and
  - determine optimal factor settings.

https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

EDA Examples




```
sns.FacetGrid(df, hue="species_short", height=5).map(plt.scatter,
"culmen_length_mm", "culmen_depth_mm").add_legend()
```

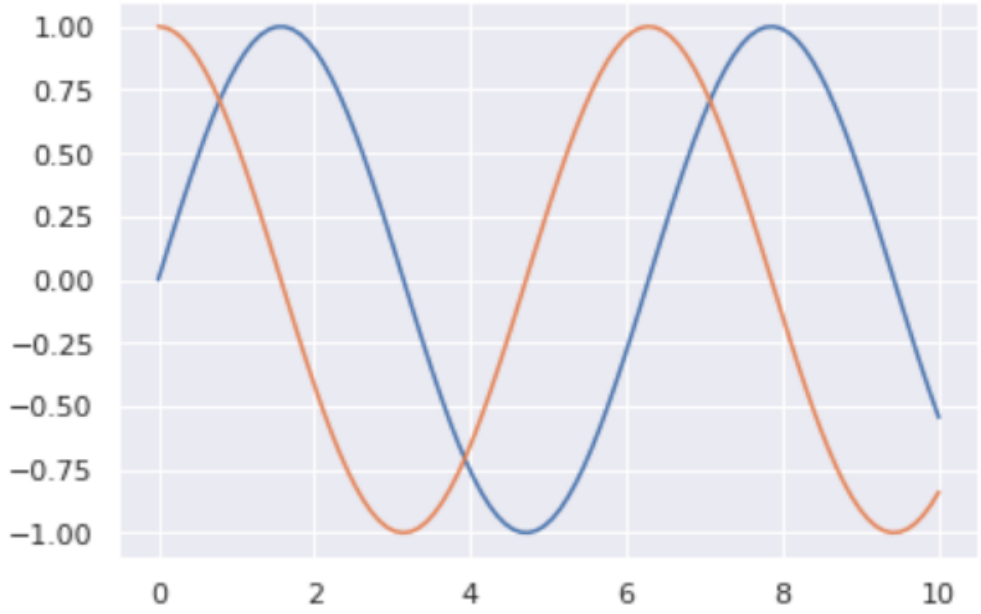



```
sns.pairplot(df, hue="species_short")
```

Visualization with Matplotlib

- import matplotlib as mpl
- import matplotlib.pyplot as plt
- plt interface is most often used
- Set "classic style": plt.style.use('classic')
- plt.show() command should be used only once per Python session.
- Can save a figure using the savefig(): fig.savefig('my_figure.png').
- %matplotlib notebook: for interactive plots
- %matplotlib inline: for static images

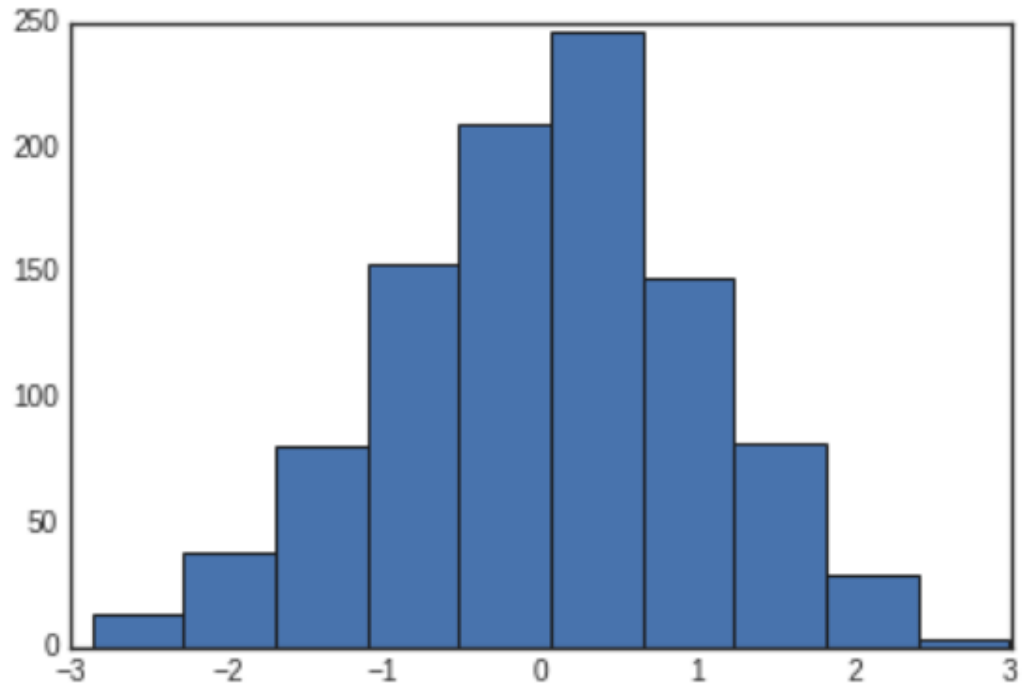
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```



Visualization and Histogram

- plt.plot is more efficient than plt.scatter.
- plt.scatter can be used to create plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.
- Histogram can be a great first step in understanding a dataset

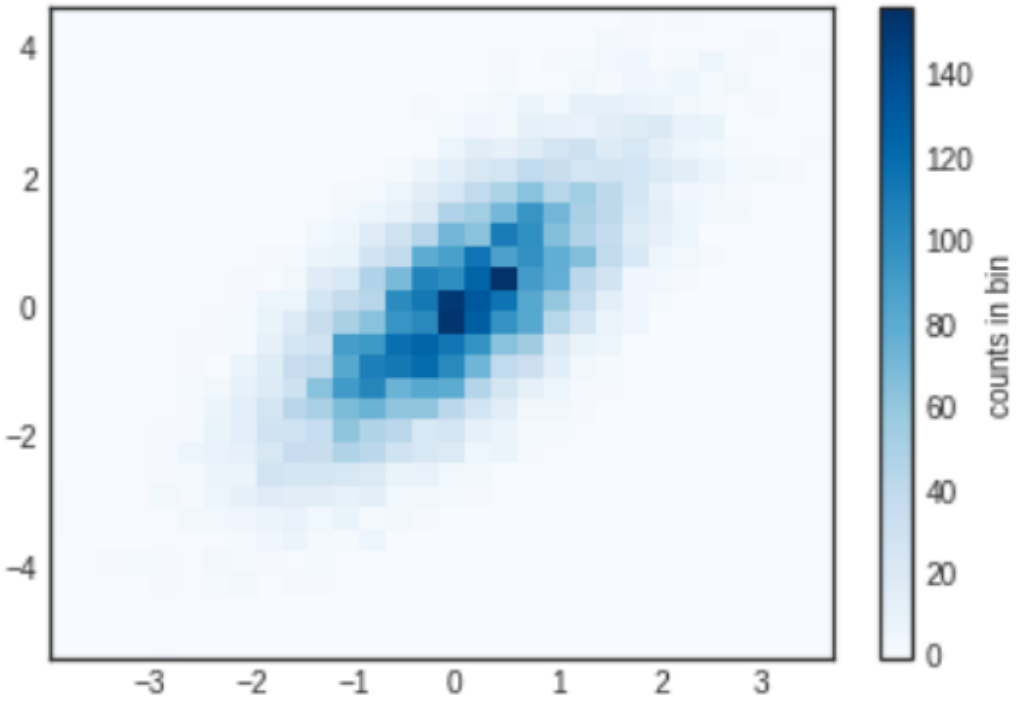
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
In[2]: plt.hist(data);
```



2D Histogram

- We created 1D-histogram by dividing the number line into bins.
- Can create 2D by dividing points among 2D bins.
  - Use Matplotlib's plt.hist2d function.

```
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean,
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```



Skewness

- Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. [Wikipedia*].
- dataframe.skew() returns unbiased skew over requested axis (of DataFrame object) Normalized by N-1.
- Output 0 denotes a symmetrical distribution.

```
import pandas as pd
dataVal = [(10,20,30,40,50,60,70),
           (10,10,40,40,50,60,70),
           (10,20,30,50,50,60,80)]

dataFrame = pd.DataFrame(data=dataVal);
print("DataFrame:")
print(dataFrame)

skewValue = dataFrame.skew(axis=1)
print("Skew:")
print(skewValue)
```

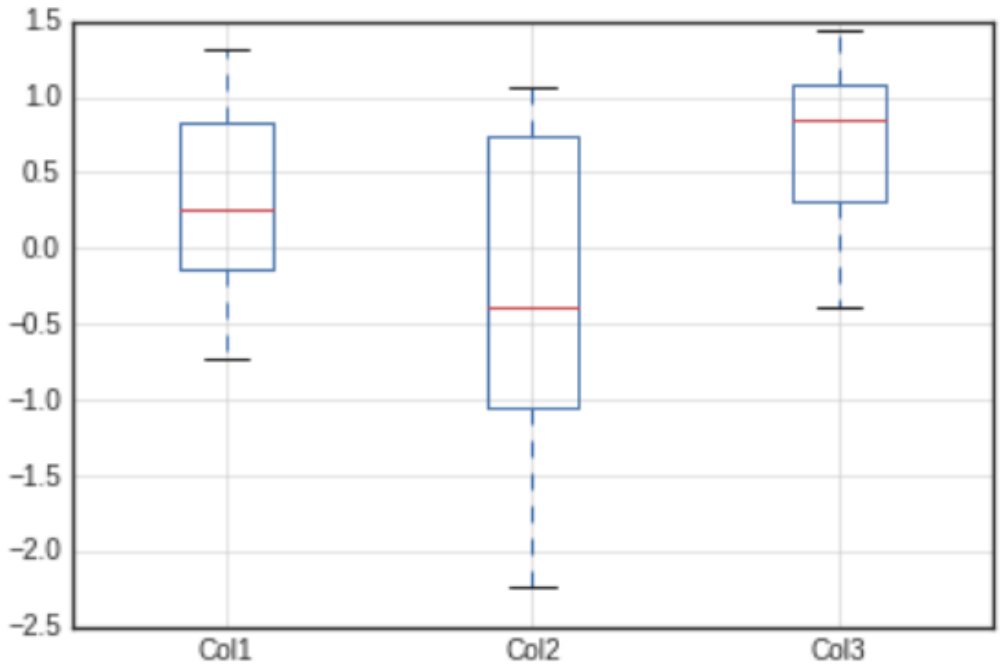
```
DataFrame:
    0   1   2   3   4   5   6
0  10  20  30  40  50  60  70
1  10  10  40  40  50  60  70
2  10  20  30  50  50  60  80
Skew:
0    0.000000
1   -0.340998
2    0.121467
dtype: float64
```

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

Box Plot

- A box plot is a method for graphically depicting groups of numerical data through their quartiles.

```
np.random.seed(1234)
df = pd.DataFrame(np.random.randn(10, 4),
                  columns=['Col1', 'Col2', 'Col3', 'Col4'])
boxplot = df.boxplot(column=['Col1', 'Col2', 'Col3'])
```

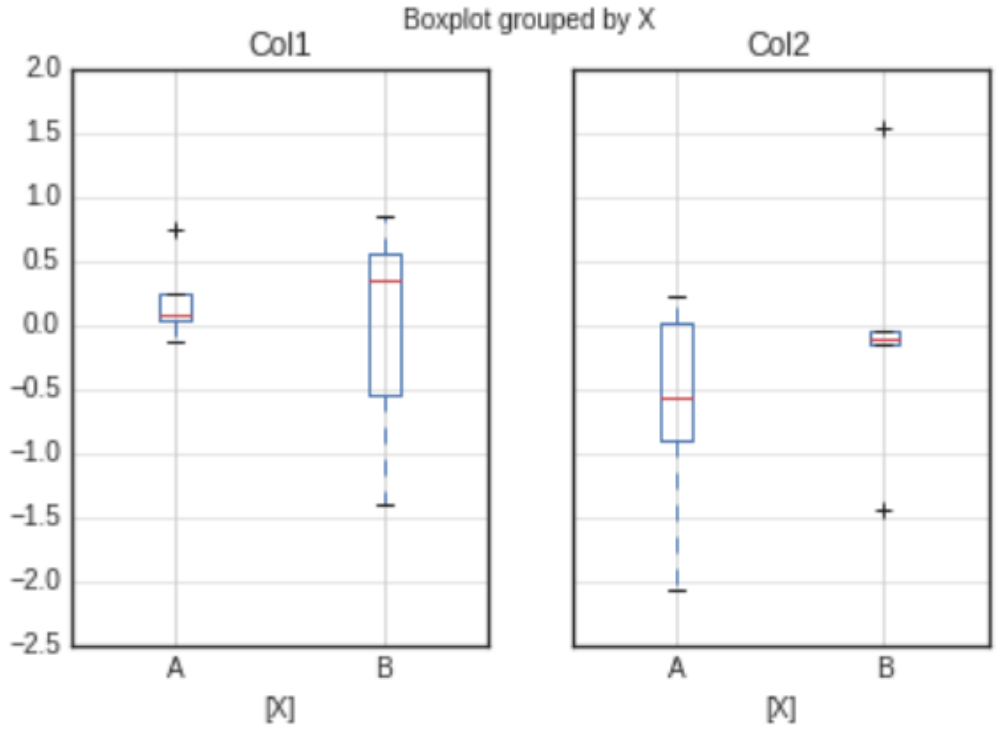


https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html

Box Plot

- Boxplots of variables distributions grouped by the values of a third variable can be created using the option **by**.

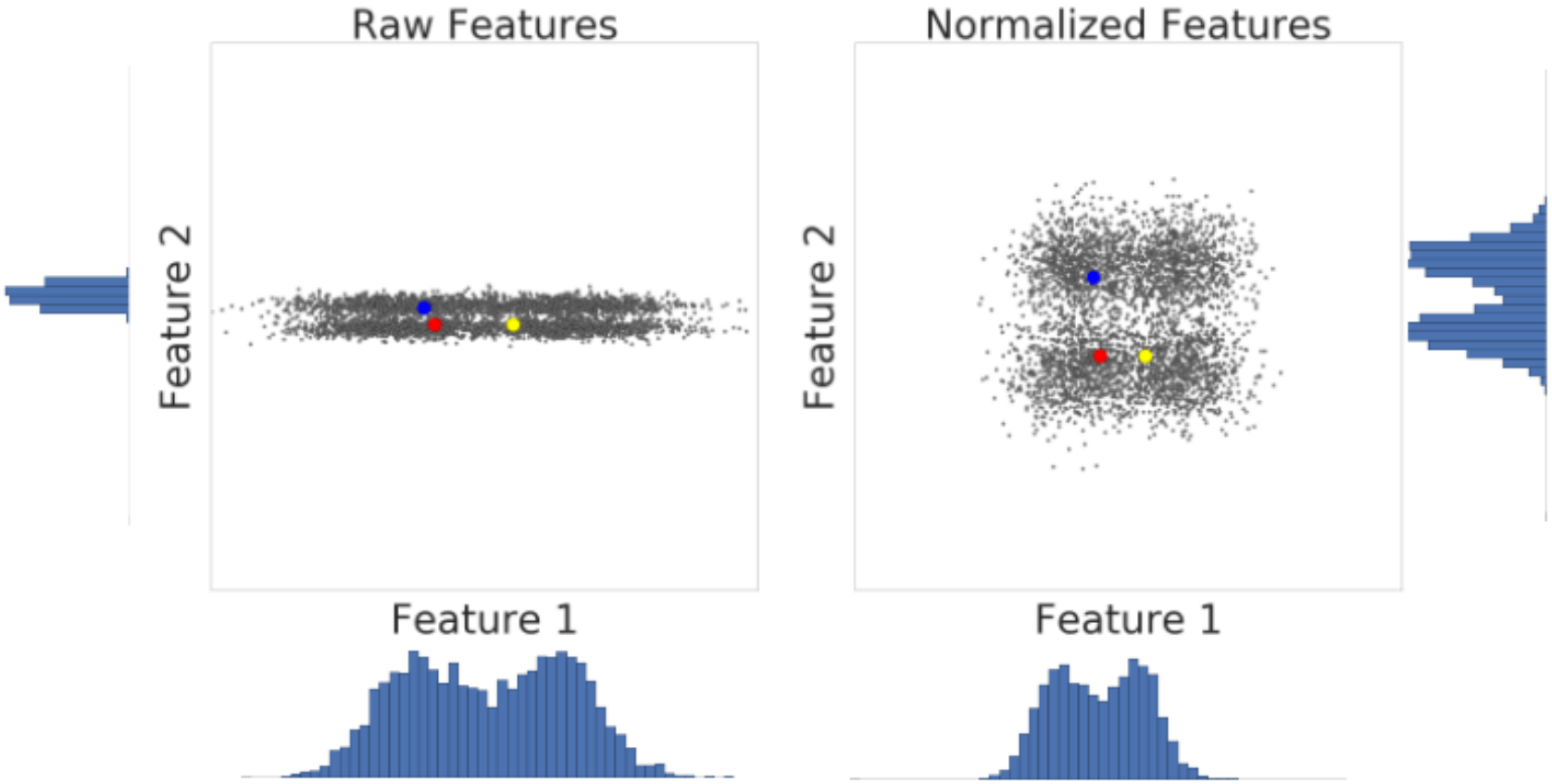
```
df = pd.DataFrame(np.random.randn(10, 2),
                  columns=['Col1', 'Col2'])
df['X'] = pd.Series(['A', 'A', 'A', 'A', 'A',
                     'B', 'B', 'B', 'B', 'B'])
boxplot = df.boxplot(by='X')
```



Scaling and Normalizing Data

- Some learning algorithms are sensitive to the scale differences in variables
- E.g. one variable is in the range of 0-1 while another 100-10000
- To mitigate such issues data can be
  - scaled (e.g. from 0 to 1) sklearn.preprocessing.MinMaxScaler()
  - standardized (data points are expressed as σ from mean) sklearn.preprocessing.StandardScaler()
  - log transformed

Example of normalization



sklearn.preprocessing.StandardScaler()

Image from https://developers.google.com/machine-learning/clustering/prepare-data

Pipelines

- Data transformation steps needed to be executed multiple times in right order.
    - Pipeline class from Scikit-Learn.
- Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps
- All but the last estimator must be transformers (i.e., they must have a fit_transform() method)

```
Pipeline([('imputer', SimpleImputer()),('scaler', StandardScaler())])
```

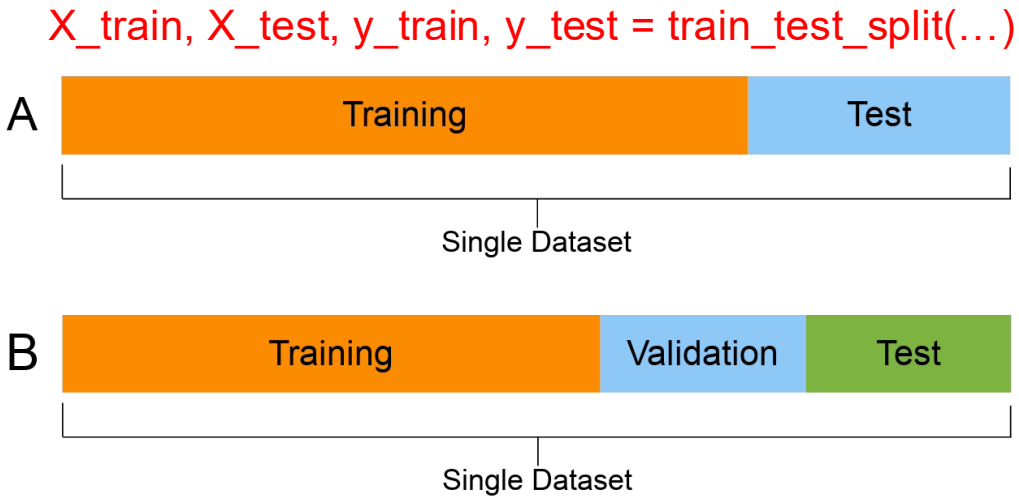
Train, validation and test sets

Datasets can be divided into 2 or 3 subsets:

- Training set, to train the model
- Test set, to confirm that the model works

Or

- Training set, to train the model
- Validation set, to tune the hyperparameters
- Test set, to confirm that the model works



Why use a validation set?

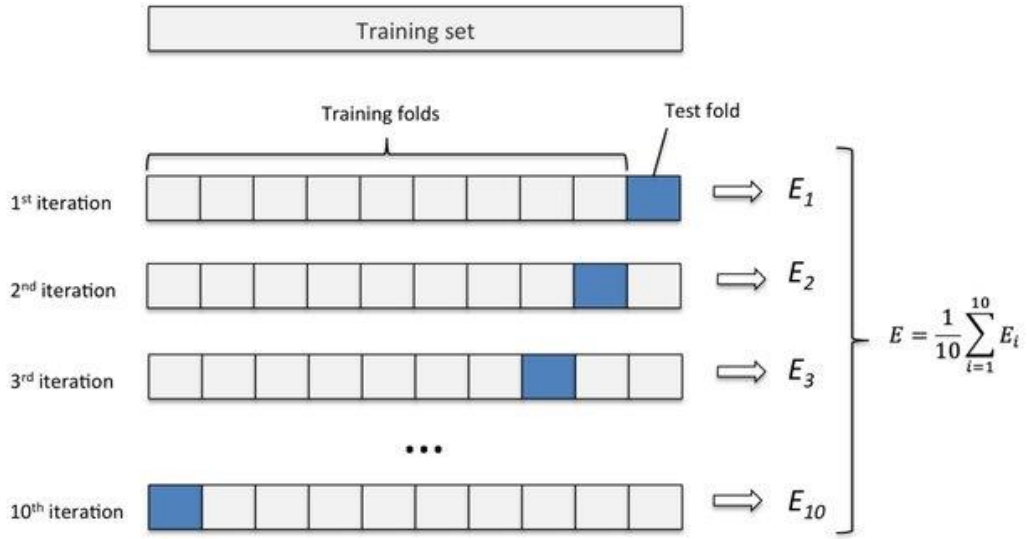
- Fine-tuning a ML model is an iterative process where you want to adjust the algorithm to perform better so that it performs well on both the training data and test set.

- To ensure your model can generalize, it is recommended to evaluate the model on a validation set instead of the test set during hyperparameter tuning.

- The test set (aka holdout set) is used only on the final model to get an unbiased evaluation of the model's performance and to ensure you haven't overfitted the model to perform well on the testing data.

Cross-validation

- A less biased or less optimistic estimate of the model than train/test split.
- Typically implemented as K-folds cross validation, where k is the number of splits.



KFold(n_splits=10, random_state=42, shuffle=False)

Image from Introduction to Support Vector Machines and Kernel Methods, Ashfaque and Iqbal, 2019

Accuracy, Recall and Precision

- Accuracy is the proportion of all classifications that were correct, whether positive or negative. Use as a rough indicator of model training progress/convergence for balanced datasets.

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Recall or true positive rate (TPR) is the proportion of all actual positives that were classified correctly as positives.

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

- Precision is the proportion of all the model's positive classifications that are actually positive.

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

TP is the number of true positives
FN is the number of false negatives
FP is the number of false positives

Do not use Recall & Precision in an imbalanced dataset where the number of actual positives is very, very low.

F1 Score

- F1 score/ balanced F-score / F-measure: is a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.
- Relative contribution of precision and recall to F1 score are equal.
- F1 is by default calculated as 0. [0 means no true positives, false negatives, or false positives].
- It balances the importance of precision and recall, and is preferable to accuracy for class-imbalanced datasets.

$$F_1 = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

Confusion Matrix

- Confusion matrix evaluate the accuracy of a classification.
  - Rows represent the actual classes the outcomes should have been.
  - Columns represent the predictions we have made.
    - True Positives (TP): Number of instances correctly predicted as positive.
    - True Negatives (TN): Number of instances correctly predicted as negative.
    - False Positives (FP): Number of instances incorrectly predicted as positive.
    - False Negatives (FN): Number of instances incorrectly predicted as negative.

		Predicted	
		Negative *(class 0)*	**Positive** *(class 1)*
Actual	**Negative** *(class 0)*	True negative TN	False positive FP
	Positive *(class 1)*	False negative FN	True positive TP

Recall

Precision

Conclusion

- Managing data not easy.
- Need to perform pre-processing before applying model.
- Also need to know the data structure via EDA.
- After knowing data apply model(s).
- Next lecture will focus on unsupervised learnings.

Resources

- Documentation and tutorial for using seaborn in plotting: https://seaborn.pydata.org/introduction.html
- Cheat sheet for scikit-learn: https://datacamp-community-prod.s3.amazonaws.com/5433fa18-9f43-44cc-b228-74672efcd116
- Read more about transforming data: https://developers.google.com/machine-learning/clustering/prepare-data
- Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.