Week 15. Data Quality Challenges - Part I

Hwanjun Song
KAIST, Dept. of ISE

Two Different Perspective of Data Quality

- **Part I: Conventional data quality**
  - Data quality is defined by the accuracy, consistency, and reliability on data values themselves
    - E.g., some values in datasets are outlier and missing
    - E.g., some values in datasets have incorrect formats and inconsistency

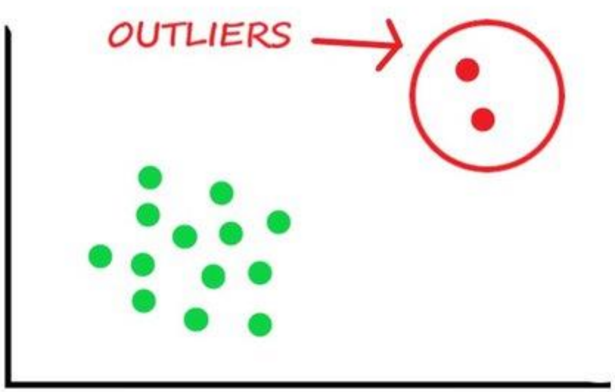
- **Part II: Modern data quality** (connected with ML/DL)
  - Data quality is defined as ensuring that the AI model performs as expected by the data without negatively affecting its performance
    - E.g., Class imbalance, noisy labels drops the test accuracy of AI models
    - E.g., Insufficient data causes the overfitting of the AI models

Today's Contents

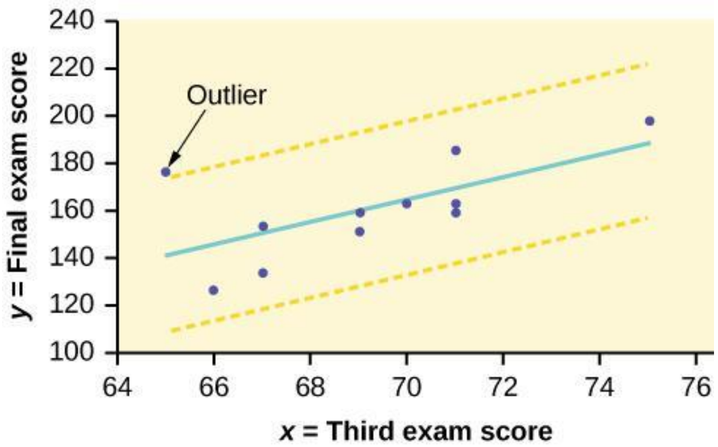
- **Outliers**
  - Quartile Analysis
  - Clustering-based Outlier Detection
- Missing Values
  - Data Deletion
  - Simple and Advanced Data Imputation
- Other conventional quality issues
  - Duplicated data, Incorrect format, Inconsistent data

What is Outlier?

- An outlier is an observation that appears to **deviate so much** from other data points of the dataset
  - E.g., In a dataset of monthly temperatures for a city where most values range from 20°C to 35°C, a recorded temperature of -10°C
  - E.g., In a dataset measuring the time it takes runners to complete a marathon, where most times are between 3 and 6 hours, a recorded time of 15 minutes



Multi-variate data (2-dim)



Correlation between two

What Make Outliers – error and no-error

- **Human errors**, which occurs when incorrect data manually entered (e.g., human mistakenly adds wrong entry to the table)
- **Instrument/measurement error**, which arises when a device or sensor malfunctions (e.g., a fault thermometer recording 150°C in normal weather)
- **Data processing error**, which happens during data cleaning or transformation processes (e.g., accidentally multiplying instead of dividing)
- **Sampling error**, which occur when data is collected from an inappropriate population (e.g., extracting data from wrong/unreliable/biased sources)

- **Not an error**: the value is just extreme, a "novel" or "abnormal" in the data
  - E.g., fraud in credit card history, health issue in healthcare monitoring, stock market anomaly – a sudden price jump

What Problem it Makes?

- **Skewing statistical measures**: outliers can significantly impact statistical measures, such as mean and standard deviation
  - E.g., [1, 3, 4], mean = 3 vs [1, 3, 4, 100], mean = 54

- **Reducing interpretability**: outliers can obscure meaningful patterns in the data and make it difficult to interpret the results

- **Reducing Robustness**: outliers make an AI model to predict the outliers as normal data, due to the overfitting

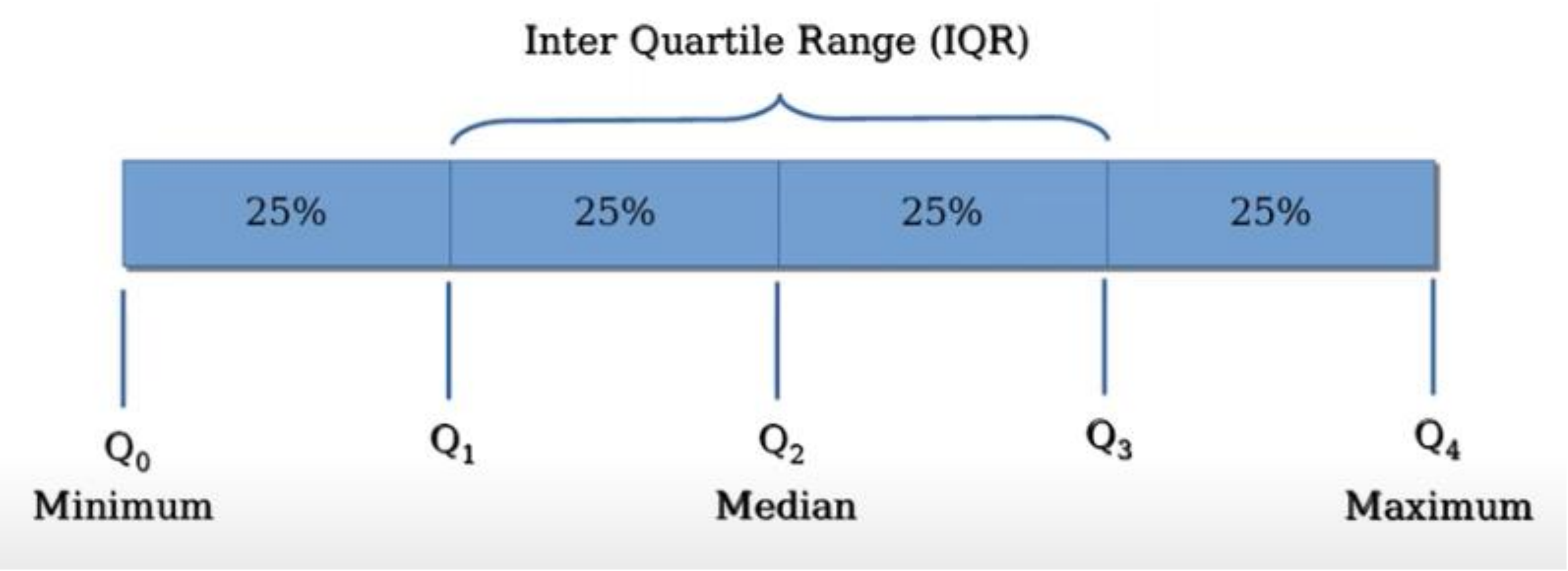
How to Solve the Issue?

- **Outlier Detection:** A task of identifying data points that deviate significantly from the majority of data
- **Challenge**: The exact number of outliers and their range of values across features are unknown, making detection more complex
- **Assumption**: There are considerably more “normal” data points than ”abnormal” data points in out dataset

- Representative Methods:
  - Quartile-based Detection
  - Clustering-based Detection
  - etc

Quartile-based Outlier Detection

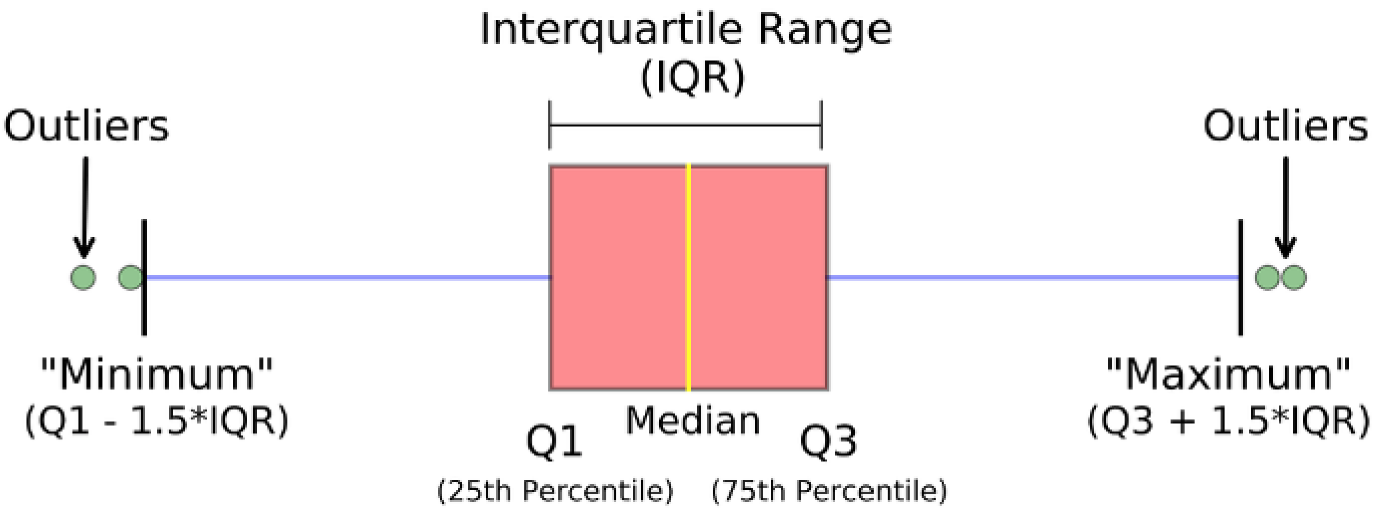
- A **quartile** is a statistical term that divides a dataset into **four equal parts**, each containing 25% of the data points
  - Five number summary: [Min (Q0), Q1, Q2, Q3, Max (Q4)]
  - Inter-quartile range: IQR = Q3 – Q1
    - It provides a measure of its spread while excuding outliers.



Quartile-based Outlier Detection

- Let's use IQR to define Outliers
  - Use 1.5 x IQR (Q3-Q1) to set the max/min boundary of the values in the dataset

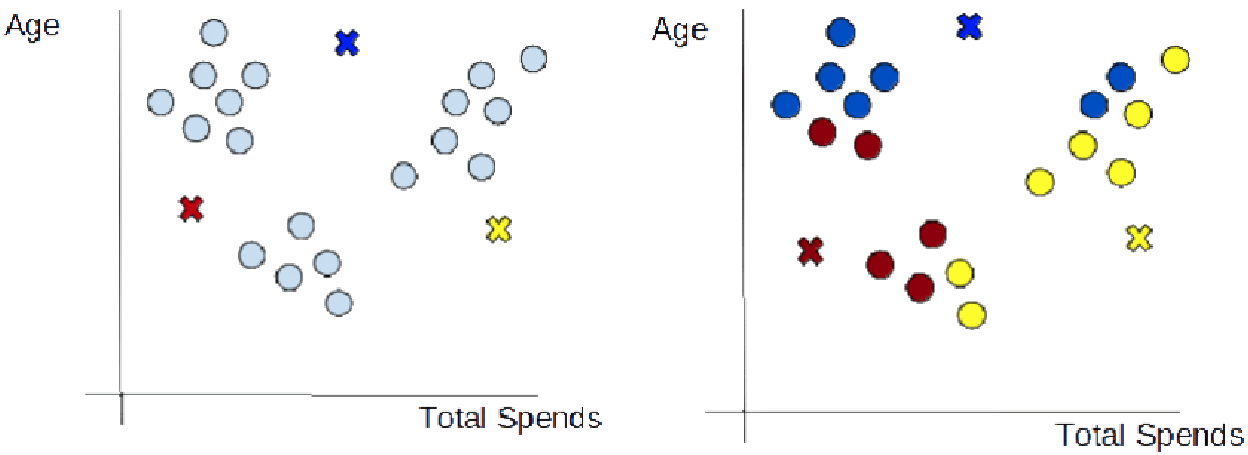
- Min Boundary: Q1 – 1.5 x IQR
- Max Boundary: Q3 + 1.5 x IQR



Clustering-based Detection

- [Recap] Clustering is a technique used to group similar data points based on their neighborhood (i.e., distance between data points)

- k-Means Clustering
  - Initializes centroids (k center points)
  - Assigns data points to the nearest centroids using Euclidean distance

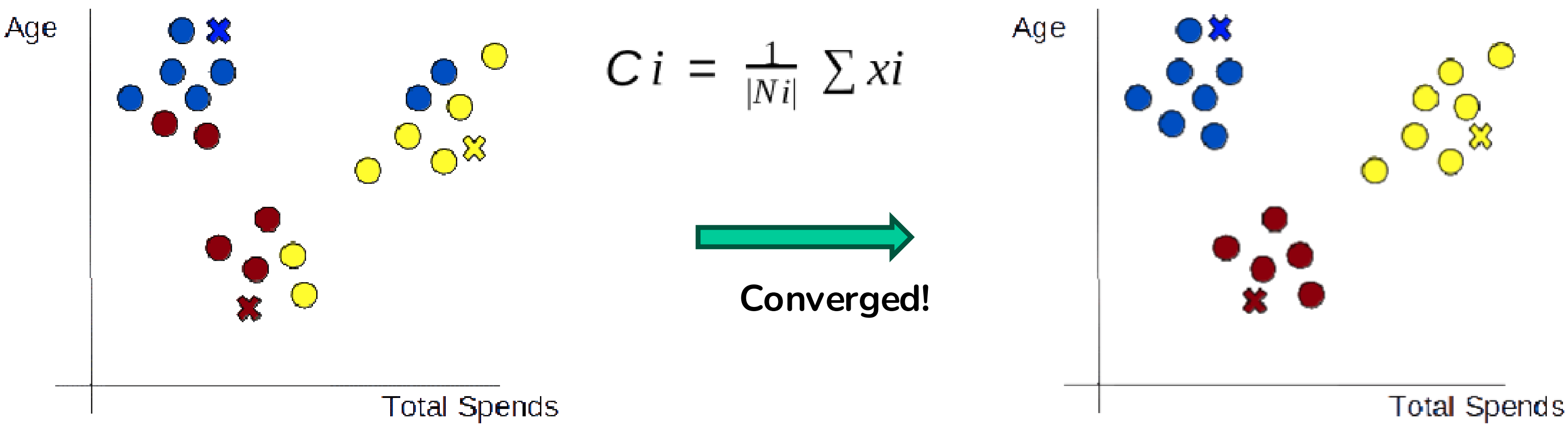


$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Clustering-based Detection

- [Recap] Clustering is a technique used to group similar data points based on their neighborhood (i.e., distance between data points)

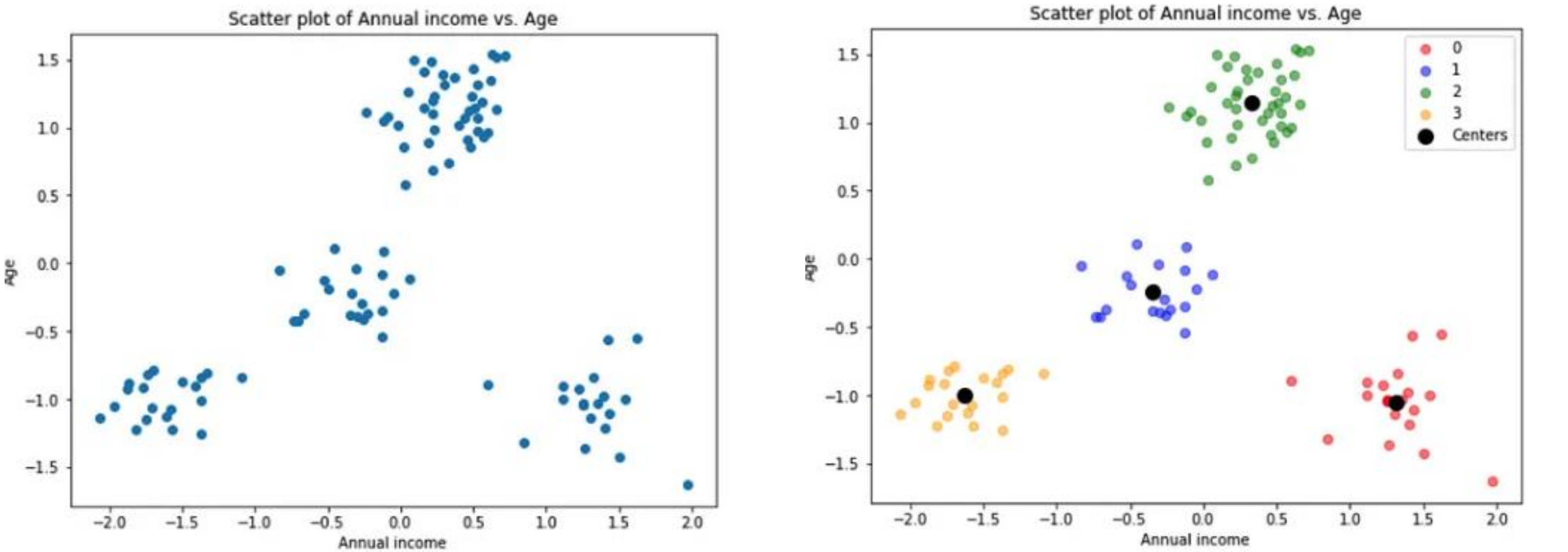
- k-Means Clustering
  - Updates centroids using the mean of data points inside the update clusters
  - Repeat the assignment and update of the centroids until converges



$$C_i = \frac{1}{|N_i|} \sum x_i$$

Clustering-based Detection

- Outlier detection using k-Means
  - Step 1. Running k-Means clustering on our datasets

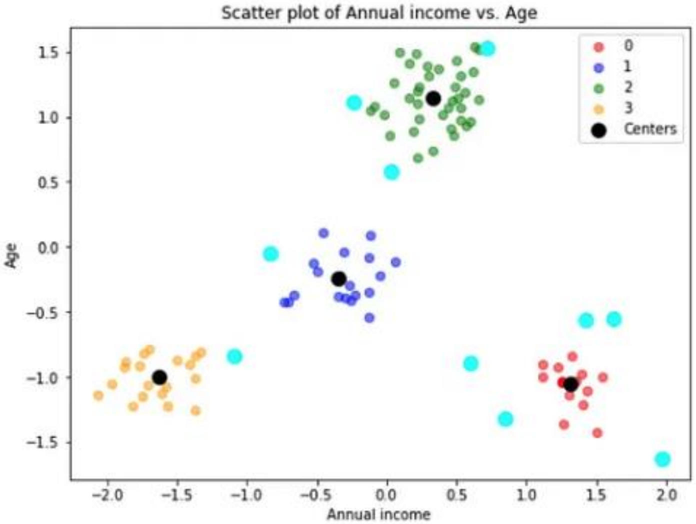


Clustering result

Clustering-based Detection

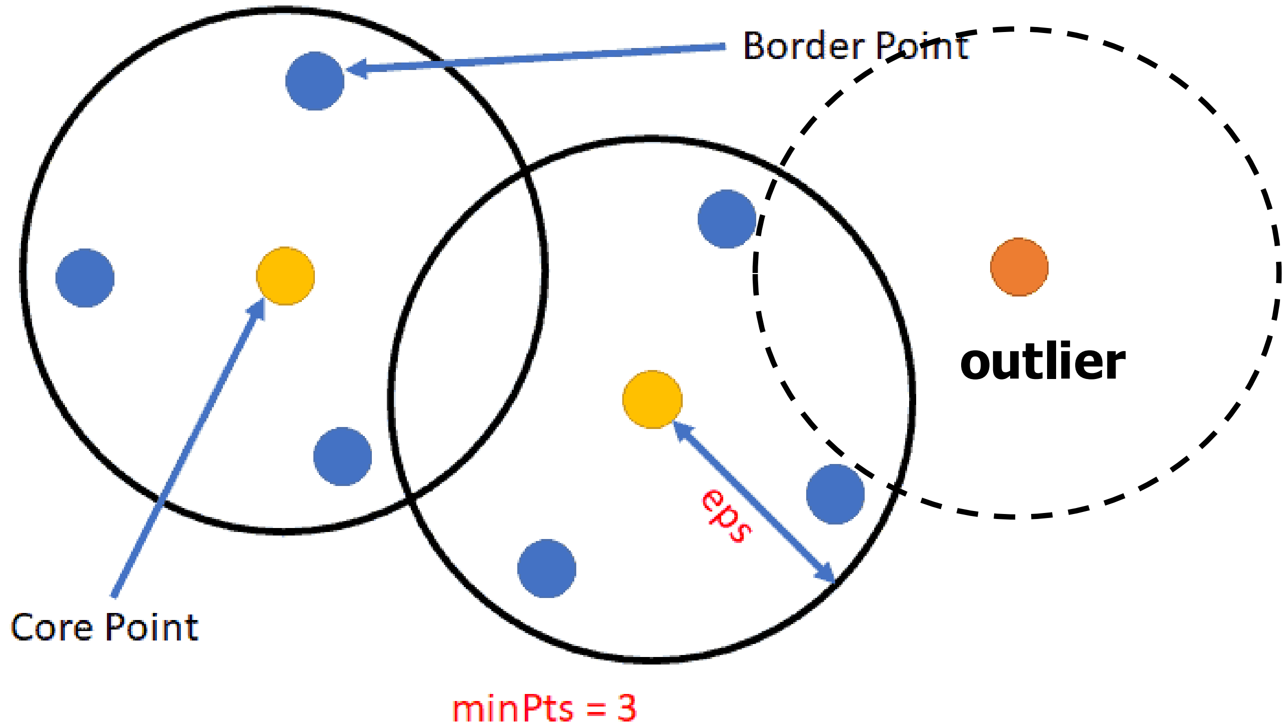
- Outlier detection using k-Means
  - Step 2. Computes the distance from each centroid to data points inside its cluster
  - Step 3. Sorts the distance by descending order from farthest to closet
  - Step 4. Sets the maximum distance (or top-k) to define outliers

	Income	Age	label	distance
5	0.600351	-0.896490	0	0.726
9	1.428157	-0.557522	0	0.507
26	1.967119	-1.629821	0	0.876
28	-1.091500	-0.838598	3	0.556
54	0.721616	1.524822	2	0.546
62	0.029044	0.580838	2	0.638
67	-0.829371	-0.054868	1	0.517
78	-0.232804	1.112968	2	0.564
80	0.847198	-1.316357	0	0.534
95	1.621387	-0.552568	0	0.587



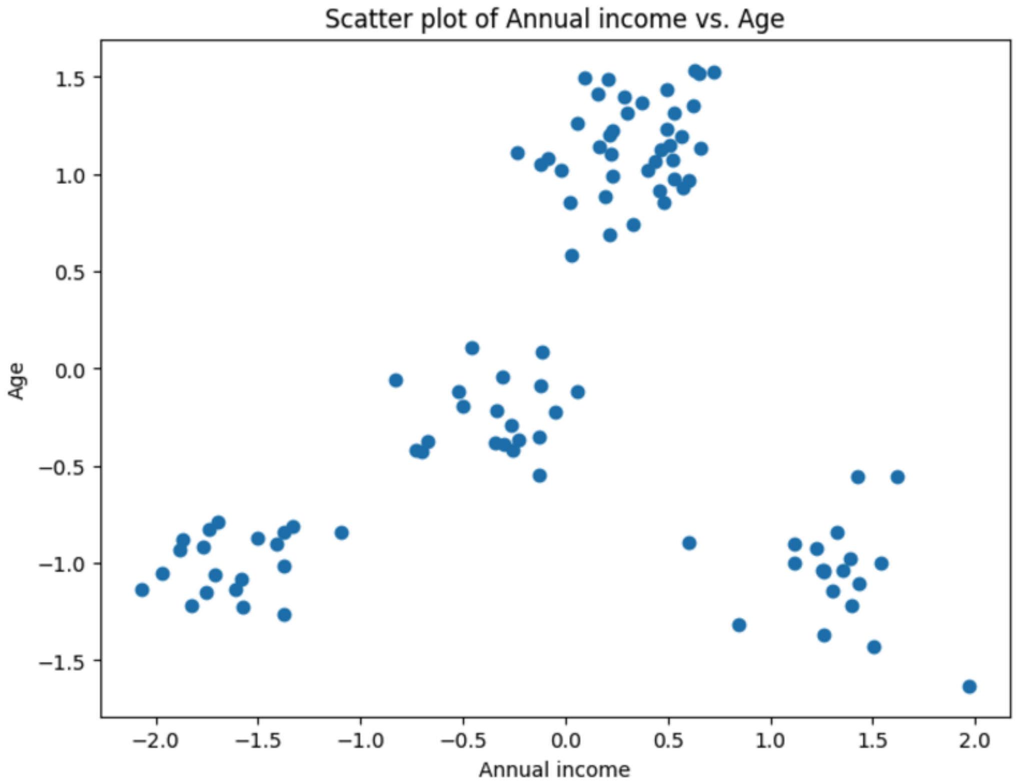
Other Clustering?

- **DBSCAN** is a density-based clustering algorithm that expand small clusters to larger clusters by using the "density reachable/connected" concept.
- Recall that:
  - Outlier: A point p is a outlier (noise) point if p is neither a core point nor a border point



k-Means based Outlier Detection in Python

- Given a dataset: the relationship btw. ”age” and “annual income”
- Let’s find some outlier data points

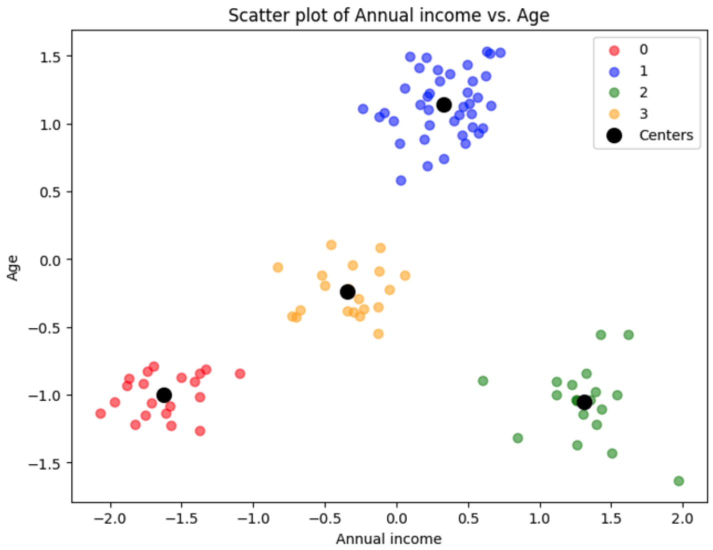


k-Means based Outlier Detection in Python

- Fitting k-Means on the dataset

```
# K-means clustering
km = KMeans(n_clusters=4)
model = km.fit(customer)
```

- Compute the distance from centers




```
customer['label'] = model.labels_
customer['distance'] = distance_from_center(customer.Income, customer.Age, customer.label)
```

```
# Create new columns: label, distance
def distance_from_center(income, age, label):
    '''
    Calculate the Euclidean distance between a data point and the center of its cluster.
    :param float income: the standardized income of the data point
    :param float age: the standardized age of the data point
    :param int label: the label of the cluster
    :rtype: float
    :return: The resulting Euclidean distance
    '''
    center_income =  model.cluster_centers_[label,0]
    center_age =  model.cluster_centers_[label,1]
    distance = np.sqrt((income - center_income) ** 2 + (age - center_age) ** 2)
    return np.round(distance, 3)
```

k-Means based Outlier Detection in Python

- Detecting outliers
  - Let's use "selecting top-k farthest points" as the criterion for our outliers!
  - You can introduce other criteria to define outliers (e.g., distance)

```
# Find outliers: top-10 farthest data points from their centroids
outliers_idx = list(customer.sort_values('distance', ascending=False).head(10).index)
outliers = customer[customer.index.isin(outliers_idx)]
```

Today's Contents

- Outliers
  - Quartile Analysis
  - Clustering-based Outlier Detection
- **Missing Values**
  - Data Deletion
  - Simple and Advanced Data Imputation
- Other conventional quality issues
  - Duplicated data, Incorrect format, Inconsistent data

What is Missing Values?

- Missing values refer to the absence of data points for particular features (attributes) in a dataset
- Dealing with missing values involves strategies like deletion or imputation to ensure data completeness for analysis

Row no	State	Salary	Yrs of Experience
1	NY	57400	Mid
2	TX		Entry
3	NJ	90000	High
4	VT	36900	Entry
5	TX		Mid
6	CA	76600	High
7	NY	85000	High
8	CA		Entry
9	CT	45000	Entry

Missing values

What Make Missing Values?

- **Non-response:** In data collection, respondents may choose not to answer certain questions, resulting in missing values
- **Data Transformation**: Missing values can occur if the data merging or data transformation processes are not handled properly
- **Data Loss**: Data can be lost due to technical issues, corruption, or accidental deletion
- **Privacy Concerns**: Data can be missing due to privacy concerns or legal restrictions, where certain information cannot be disclosed
- **Equipment Malfunction**: Data collection can involves sensors or automated systems failure, leading to missing data values
- And others, etc

What Problems it Makes?

- **Bias in Analysis:** missing values can introduce bias into statistical analysis and machine learning models
- **Increased uncertainty:** missing values make analysis harder to draw confident conclusion from the data
- **Data Imbalance:** Missing values in specific features create an imbalance, leading to skewed results and misrepresentation of the data distribution.

Easy Solution: Deletion of Missing Values

- **Deletion** is removing rows or columns from the dataset containing missing values

One	Two
0	2
1	3
2	0
NaN	1

One	Two
0	2
1	3
2	0

Row-wise deletion

One	Two
0	2
1	3
2	0
NaN	1

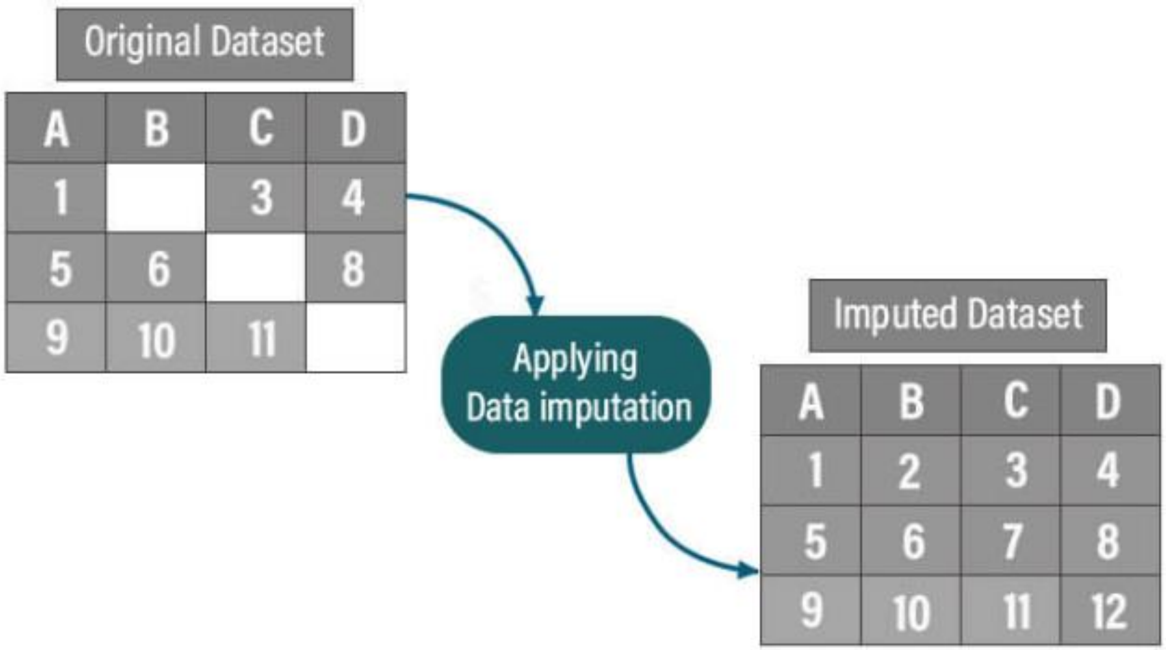
Two
2
3
0
1

Column-wise deletion

- While straightforward, this approach can lead to information loss and biased results if the missingness is not random
  - So, not recommended in real-world scenrios

Better Approach: Data Imputation (대치)

- **Data imputation** is to estimate missing values in data based on observed data points, allowing for the completion datasets



- Various methods are employed to fill in missing values and maintain the integrity of the data for analysis

Simple Imputation: Mean, Median, Mode Imputation

- Mean/Median/Mode imputation is simple method for handling missing data by replaying missing values with the mean/median/mode of the observed values for that attribute
- We can preserve the overall statistical distribution of the data, but it may reduce the variability of real values

	Age	Gender	Fitness_Score
0	20	M	NaN
1	25	F	7.0
2	30	M	NaN
3	35	M	7.0
4	36	F	6.0
5	42	F	5.0
6	49	M	6.0
7	50	F	4.0
8	55	M	4.0
9	60	F	5.0
10	66	M	4.0
11	70	F	NaN
12	75	M	3.0
13	78	F	NaN

Mean Imputed

	Age	Gender	Fitness_Score
0	20	M	5.1
1	25	F	7.0
2	30	M	5.1
3	35	M	7.0
4	36	F	6.0
5	42	F	5.0
6	49	M	6.0
7	50	F	4.0
8	55	M	4.0
9	60	F	5.0
10	66	M	4.0
11	70	F	5.1
12	75	M	3.0
13	78	F	5.1

Advanced Imputation 1: kNN Imputation

- kNN (k-nearest neighbor) imputation identifies the k closest (similar) data points w.r.t other observed attributes
- And, then, replace the messing value of the data point with the mean of its k-nearest neighbors

#	A	B	C
1	N/A	3	4
2	1	12	11
3	7	4	3
4	1	7	10
5	8	4	4

observed attributes

2-NN

#	A	B	C
1	7.5	3	4
2	1	12	11
3	7	4	3
4	1	7	10
5	8	4	4

Advanced Imputation 2: Iterative Imputation

- A way of iteratively update the missing values by conducting **regression task** of predicting the missing value

age	experience	salary(K)
25		50
27	3	
29	5	80
31	7	90
33	9	100
	11	130

(1) Mean imputation for initialization

age	experience	salary(K)
25	7	50
27	3	90
29	5	80
31	7	90
33	9	100
29	11	130

(2) Regression using observed values

age	experience	salary(K)
25	6.7	50
27	3	97.2
29	5	80
31	7	90
33	9	100
26.5	11	130

(3) Repeat regression and update until converge

age	experience	salary(K)
25	6.1	50
27	3	96.2
29	5	80
31	7	90
33	9	100
25.7	11	130

Handling Missing Values in Python

- Dataset: Pima-Indians-diabetes dataset
  - Nine attributes: Pregnancies (과거임신횟수), **Glucose** (포도당농도), **BloodPressure** (이완기 혈압), **SkinThickness** (피하지방두께), **Insulin** (인슐린 농도), **BMI** (체질량지수), DiabetsPredigreeFuncion (당뇨 가족력 계수), Age (나이,), Outcome (당뇨병 여부)
  - Attributes with index 1-5 (in bold) includes many missing values (marked "0")

```python
# load the dataset and review rows
from pandas import read_csv

# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)

dataset.head(10)
```

	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Handling Missing Values in Python

- Count the number of missing value
  - Attributes with index 1-5 (in bold) includes many missing values (marked "0")
  - Count the # of data points with "0" value for each attribute 1-5

```
# summarizing the number of missing values for each variable
num_missing = (dataset[[1,2,3,4,5,]] == 0).sum()
# report the results
print(num_missing)
```

```
1      5
2     35
3    227
4    374
5     11
dtype: int64
```

Handling Missing Values in Python

- Approach 1. Row-wise Deletion of Missing Values
  - Replaces “0” with ”nan” and then use dropna() in Pandas
  - But, this erased too many data points…

```
# example of removing rows that contain missing values
from numpy import nan
from pandas import read_csv

# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)

# summarize the shape of the raw data
print('original:', dataset.shape)

# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)

# drop rows with missing values
dataset.dropna(inplace=True)

# summarize the shape of the data with missing rows removed
print('after deletion:',dataset.shape)

# print the first 20 rows of data, where missing values are marked as '0'
print(dataset.head(20))
```

```
original: (768, 9)
after deletion: (392, 9)
```

Handling Missing Values in Python

- Approach 2. Imputation of Missing Values using Means

Mean Imputation

The example below uses the SimpleImputer class to replace missing values with the mean of each column then prints the number of NaN values in the transformed matrix.

Also, the number of rows in the data remains the same even after imputation.

```
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)

# retrieve the numpy array
values = dataset.values

# define the imputer
imputer = SimpleImputer(missing_values=nan, strategy='mean')

# transform the dataset
transformed_values = imputer.fit_transform(values)

# count the number of NaN values in each column
print(f'Missing: {isnan(transformed_values).sum()}')

# summarize the shape of the data after imputation
print('after deletion:',transformed_values.shape)

# print the first 20 rows of data, where missing values are marked as '0'
dataset = pd.DataFrame(transformed_values)
print(dataset.head(20))
```

```
original: (768, 9)
Missing: 0
after deletion: (768, 9)
       0      1          2         3           4          5      6     7    8
0    6.0  148.0  72.000000  35.00000  155.548223  33.600000  0.627  50.0  1.0
1    1.0   85.0  66.000000  29.00000  155.548223  26.600000  0.351  31.0  0.0
2    8.0  183.0  64.000000  29.15342  155.548223  23.300000  0.672  32.0  1.0
3    1.0   89.0  66.000000  23.00000   94.000000  28.100000  0.167  21.0  0.0
4    0.0  137.0  40.000000  35.00000  168.000000  43.100000  2.288  33.0  1.0
5    5.0  116.0  74.000000  29.15342  155.548223  25.600000  0.201  30.0  0.0
6    3.0   78.0  50.000000  32.00000   88.000000  31.000000  0.248  26.0  1.0
7   10.0  115.0  72.405184  29.15342  155.548223  35.300000  0.134  29.0  0.0
8    2.0  197.0  70.000000  45.00000  543.000000  30.500000  0.158  53.0  1.0
9    8.0  125.0  96.000000  29.15342  155.548223  32.457464  0.232  54.0  1.0
10   4.0  110.0  92.000000  29.15342  155.548223  37.600000  0.191  30.0  0.0
```

Handling Missing Values in Python

- Approach 3. kNN Imputation

KNN or K nearest neighbor imputation is yet another technique to handle missing values. You can use scikit-learn's KNNImputer to perform this imputation.

For a data point with missing values, this technique identifies the K closest points under a chosen distance metric (Euclidean by default). The number of closest points or neighbors is specified by the n_neighbors parameter. By default, the 5 closest neighbors are considered.

```python
# example of imputing missing values using KNN imputer
from numpy import nan
from numpy import isnan
from pandas import read_csv
from sklearn.impute import KNNImputer
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# retrieve the numpy array
values = dataset.values
# define the imputer
imputer = KNNImputer(n_neighbors=4)
# transform the dataset
transformed_values = imputer.fit_transform(values)
# count the number of NaN values in each column
print(f'Missing: {isnan(transformed_values).sum()}')

# print the first 20 rows of data, where missing values are marked as '0'
dataset = pd.DataFrame(transformed_values)
print(dataset.head(20))
```

Python

Missing: 0

	0	1	2	3	4	5	6	7	8
0	6.0	148.0	72.0	35.00	179.50	33.600	0.627	50.0	1.0
1	1.0	85.0	66.0	29.00	61.00	26.600	0.351	31.0	0.0
2	8.0	183.0	64.0	28.75	163.75	23.300	0.672	32.0	1.0
3	1.0	89.0	66.0	23.00	94.00	28.100	0.167	21.0	0.0
4	0.0	137.0	40.0	35.00	168.00	43.100	2.288	33.0	1.0
5	5.0	116.0	74.0	20.75	106.75	25.600	0.201	30.0	0.0
6	3.0	78.0	50.0	32.00	88.00	31.000	0.248	26.0	1.0
7	10.0	115.0	73.0	36.25	141.00	35.300	0.134	29.0	0.0
8	2.0	197.0	70.0	45.00	543.00	30.500	0.158	53.0	1.0
9	8.0	125.0	96.0	25.25	169.75	33.675	0.232	54.0	1.0
10	4.0	110.0	92.0	30.75	145.75	37.600	0.191	30.0	0.0

Handling Missing Values in Python

- Approach 4. Iterative Imputation using Regression

Scikit-learn's IterativeImputer is a more sophisticated multivariate imputation technique.

The IterativeImputer predicts the missing values of a feature by modeling it as a function of other features. The imputer, therefore, predicts the missing values of a feature using the other features as predictors.

It then imputes all missing features in a round-robin fashion. This imputation continues iteratively for max_iter number of times, and is set to 10 by default.

Because the IterativeImputer feature is still experimental, you have to enable it explicitly

```
# example of imputing missing values using Iterative imputer
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)

# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)

# retrieve the numpy array
values = dataset.values

# define the imputer
imputer = IterativeImputer(random_state=0)

# transform the dataset
transformed_values = imputer.fit_transform(values)

# count the number of NaN values in each column
print(f'Missing: {isnan(transformed_values).sum()}')

# print the first 20 rows of data, where missing values are marked as '0'
dataset = pd.DataFrame(transformed_values)
print(dataset.head(20))
```

```
Missing: 0
       0      1          2          3           4          5      6     7    8
0    6.0  148.0  72.000000  35.000000  218.903553  33.600000  0.627  50.0  1.0
1    1.0   85.0  66.000000  29.000000   70.314661  26.600000  0.351  31.0  0.0
2    8.0  183.0  64.000000  21.542781  268.507178  23.300000  0.672  32.0  1.0
3    1.0   89.0  66.000000  23.000000   94.000000  28.100000  0.167  21.0  0.0
4    0.0  137.0  40.000000  35.000000  168.000000  43.100000  2.288  33.0  1.0
5    5.0  116.0  74.000000  22.078010  125.695623  25.600000  0.201  30.0  0.0
6    3.0   78.0  50.000000  32.000000   88.000000  31.000000  0.248  26.0  1.0
7   10.0  115.0  72.971094  31.565415  136.287418  35.300000  0.134  29.0  0.0
8    2.0  197.0  70.000000  45.000000  543.000000  30.500000  0.158  53.0  1.0
9    8.0  125.0  96.000000  34.062564  161.554785  35.832462  0.232  54.0  1.0
10   4.0  110.0  92.000000  33.092437  124.835344  37.600000  0.191  30.0  0.0
```

Today's Contents

- Outliers
  - Quartile Analysis
  - Clustering-based Outlier Detection
- Missing Values
  - Data Deletion
  - Simple and Advanced Data Imputation
- **Other conventional quality issues**
  - Duplicated data, Incorrect format, Inconsistent data

Other Quality Issues

- Duplicated Data
  - Data duplication refers to the existence of multiple copies of the same data, often resulting from errors or inefficiencies in data management practices

id	first_name	last_name	email
1	Carine	Schmitt	carine.schmitt@verizon.net
4	Janine	Labrune	janine.labrune@aol.com
6	Janine	Labrune	janine.labrune@aol.com
2	Jean	King	jean.king@me.com
12	Jean	King	jean.king@me.com
5	Jonas	Bergulfsen	jonas.bergulfsen@mac.com
10	Julie	Murphy	julie.murphy@yahoo.com
11	Kwai	Lee	kwai.lee@google.com
3	Peter	Ferguson	peter.ferguson@google.com
9	Roland	Keitel	roland.keitel@yahoo.com
14	Roland	Keitel	roland.keitel@yahoo.com
7	Susan	Nelson	susan.nelson@comcast.net
13	Susan	Nelson	susan.nelson@comcast.net
8	Zbyszek	Piestrzeniewicz	zbyszek.piestrzeniewicz@att.net

Other Quality Issues

- Incorrect format
  - Incorrect format refers to data that does not adhere to the expected or standardized format for a particular data type
  - E.g., Phone number format:
    - xxxxxxxxxx +x xxx-xxx-xxxx (xxx)xxx-xxxx

CustomerID	CustomerName	Email	Phone
101	John Doe	john@example.com	1234567890
102	Jane Smith	jane@example.com	+1 456-789-0123
103	John Doe	john@example.com	789-012-3456
104	Mary Johnson	mary@example.com	0123456789
105	John Doe	johndoe@gmail.com	(234)567-8901

Other Quality Issues

- Data Inconsistency
  - Data inconsistency refers to discrepancies or contradictions in the information stored within a dataset
  - E.g., there are three rows of the same customers, but have different email addresses and phone numbers

CustomerID	CustomerName	Email	Phone
101	John Doe	john@example.com	123-456-7890
102	Jane Smith	jane@example.com	456-789-0123
103	John Doe		789-012-3456
104	Mary Johnson	mary@example.com	012-345-6789
105	John Doe	johndoe@gmail.com	234-567-8901

Solved with simple solution!

- Duplicated Data
  - Utilizes the built-in-functions in Python (.drop_duplicates() in Pandas) or queries using SQL (DISTINCT or GROUP BY clause to eliminate duplicates)

- Incorrect Format
  - Implements data validation rules and standardization procedures to ensure that data adheres to predefined formats

- Inconsistent Data
  - Establishes data governance policies and procedures to enforece consistency rules and standards across datasets

Colab for Your Practice

- Colab Link:
  https://drive.google.com/file/d/1P_DVi3QGzLEoQASjslIHEdoTjR6LS-Fv/view?usp=sharing

- Contents:
  - Outliers: Quartile Analysis and Clustering-based Outlier Detection
  - Missing Values: Data Deletion and Simple and Advanced Data Imputation
  - Other conventional quality issues
    - Duplicated data, Incorrect format, Inconsistent data

- Enjoy your practice on what we learn today!
  - If you have any question, email me or our teaching assistants.

Q & A