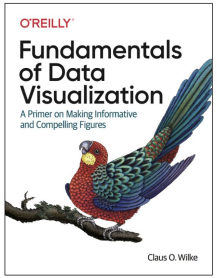
SICSS Exploratory Data Analysis and Visualization

Dr. Uzay Çetin

Fundamentals of Data Visualization

Claus O. Wilke

Welcome



This is the website for the book "Fundamentals of Data Visualization," published by O'Reilly Media, Inc. The website contains the complete author manuscript before final copy-editing and other quality control. If you would like to order an official hardcopy or ebook, you can do so at various resellers, including Amazon, Barnes and Noble, Google Play, or Powells.

The book is meant as a guide to making visualizations that accurately reflect the data, tell a story, and look professional. It has grown out of my experience of working with students and postdocs in my laboratory on thousands of data visualizations. Over the years, I have noticed that the same issues arise over and over. I have attempted to collect my accumulated knowledge from these interactions in the form of this book.

Phases of Data Science Project

- Data Requirements *(Type of data)*
- Data Collection *(Data sources)*
- Data Processing *(Data Format)*
- Data Cleaning *(Duplicates check, error check, and missing value check etc.)*
- Exploratory Data Analysis *(Understand the message contained in the data)*
- Modeling and Algorithms *(Data = Signal + Noise)*
- Data Product
- Communication

Phases of Data Science Project

- Data Requirements *(Type of data: Turkish Texts)*
- Data Collection *(Data sources: Social Media Platforms and Web)*
- Data Cleaning *(Lowercase, Remove stopwords, 512 Token etc..)*
- Exploratory Data Analysis *(Understand the message contained in the data: Clustering etc.)*
- Modeling and Algorithms *(Data = Signal + Noise, BERT based Algorithms)*
- Data Product *(Realtime Social MEdia Analysis Tool)*
- Communication *(News, advertisement etc.)*

https://summarify.io/

Data Cleaning and Preparation

- **Handling missing and incorrect/duplicate data**
- **Handling outliers**
- **Data transformation and normalization**
- **Class imbalance**

Source: Using images Effectively

Data Cleaning and Preparation

Transforming raw data into a **clean and structured format** is crucial for ensuring accurate results and for building robust and reliable models.

EDA is a process of examining the available dataset to discover patterns, spot anomalies, test hypotheses, and check assumptions using statistical measures.

Data Cleaning and Preparation

1. Handling missing values: You can either drop missing values or fill them with a default value such as the mean, median or mode. For example, using the `pandas` library in Python, you can drop missing values using `df.dropna()` or fill missing values using `df.fillna(value)`.
2. Handling outliers: Outliers can impact the results of data analysis and modeling, so it's important to detect and handle them. You can use techniques such as the Z-score method or the interquartile range (IQR) to detect outliers. For example, you can use the `zscore` function from the `scipy` library in Python to calculate the Z-score of each value in a column and detect outliers.
3. Encoding categorical variables: Categorical variables need to be transformed into numerical values for analysis and modeling. You can use one-hot encoding or label encoding to achieve this. For example, using the `pandas` library in Python, you can use `pd.get_dummies` to one-hot encode categorical variables.
4. Normalizing numerical variables: Normalizing numerical variables helps to bring them to the same scale and prevent features with larger scale from dominating others. You can use techniques such as min-max scaling or standardization to normalize the variables. For example, using the `sklearn` library in Python, you can use `MinMaxScaler` or `StandardScaler` to normalize the variables.

Handling Missing Values

```
import pandas as pd

# Load the data into a Pandas DataFrame
df = pd.read_csv('data.csv')

# Replace missing values with the mean of the column
df.fillna(df.mean(), inplace=True)

# Drop rows with missing values
df.dropna(inplace=True)
```

You can replace missing values with a default value, such as the mean, median or mode of the column, or drop the rows containing missing values.

Handling Missing Values

```
import numpy as np
import pandas as pd

# Generate a synthetic data set with 100 samples and 5 features
np.random.seed(0)
data = np.random.randn(10, 5)

cs = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']
# Convert the data to a pandas DataFrame
df = pd.DataFrame(data, columns = cs)

# Introduce missing values into the DataFrame
df.iloc[3:5, 1] = np.nan
df.iloc[6:8, 3] = np.nan

df
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	NaN	-0.205158	0.313068	-0.854096
4	-2.552990	NaN	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	NaN	-0.347912
7	0.156349	1.230291	1.202380	NaN	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

Handling Missing Values

```
# Option 1: Drop missing values
df_dropna = df.dropna()
df_dropna
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	NaN	-0.205158	0.313068	-0.854096
4	-2.552990	NaN	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	NaN	-0.347912
7	0.156349	1.230291	1.202380	NaN	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

Handling Missing Values

```
# Option 2: Fill missing values with the mean
df_fillna_mean = df.fillna(df.mean())
df_fillna_mean
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	0.223240	-0.205158	0.313068	-0.854096
4	-2.552990	0.223240	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	0.462489	-0.347912
7	0.156349	1.230291	1.202380	0.462489	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	NaN	-0.205158	0.313068	-0.854096
4	-2.552990	NaN	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	NaN	-0.347912
7	0.156349	1.230291	1.202380	NaN	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

Handling Missing Values

```
# Option 3: Fill missing values with the median
df_fillna_median = df.fillna(df.median())
df_fillna_median
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	0.389160	-0.205158	0.313068	-0.854096
4	-2.552990	0.389160	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	0.217371	-0.347912
7	0.156349	1.230291	1.202380	0.217371	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	NaN	-0.205158	0.313068	-0.854096
4	-2.552990	NaN	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	NaN	-0.347912
7	0.156349	1.230291	1.202380	NaN	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740

Mean, median, and mode

The mean is the average of all values in a dataset. It is calculated by summing all values and dividing by the number of values.

The median is the middle value of a dataset when it is sorted in ascending or descending order. It represents the "typical" value in a dataset with a symmetrical distribution.

The mode is the value that appears most frequently in a dataset. It can be used to represent the "typical" value in a dataset with a skewed distribution.

They are commonly used measures of central tendency in statistics.

Each provides a different perspective on the central location of a set of data.

Central Tendencies Mean, Median, Mode

The mean is the average of all values in a dataset. It is calculated by summing all values and dividing by the number of values.

The median is the middle value of a dataset when it is sorted in ascending or descending order. It represents the "typical" value in a dataset with a symmetrical distribution.

The mode is the value that appears most frequently in a dataset. It can be used to represent the "typical" value in a dataset with a skewed distribution.

```python
import numpy as np
import pandas as pd

# Generate synthetic data
data = [1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]

# Convert the data to a pandas Series
series = pd.Series(data)

# Calculate mean
mean = series.mean()
print(f'Mean: {mean:.2f}')

# Calculate median
median = series.median()
print(f'Median: {median:.2f}')

# Calculate mode
mode = series.mode().iloc[0]
print(f'Mode: {mode:.2f}')
```

```
Mean: 3.67
Median: 4.00
Mode: 5.00
```

Mean, median, and mode

The mean is the average of all values in a dataset. It is calculated by summing all values and dividing by the number of values.

The median is the middle value of a dataset when it is sorted in ascending or descending order. It represents the "typical" value in a dataset with a symmetrical distribution.

The mode is the value that appears most frequently in a dataset. It can be used to represent the "typical" value in a dataset with a skewed distribution.

```python
import numpy as np
import pandas as pd

# Generate synthetic data
data = [1,2,3,4]

# Convert the data to a pandas Series
series = pd.Series(data)

# Calculate mean
mean = series.mean()
print(f'Mean: {mean:.2f}')

# Calculate median
median = series.median()
print(f'Median: {median:.2f}')

# Calculate mode
mode = series.mode().iloc[0]
print(f'Mode: {mode:.2f}')
```

```
Mean: 2.50
Median: 2.50
Mode: 1.00
```

Handling Missing Values with **Linear Regression**

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

# Generate synthetic data with missing values
data = [[1, 2, np.nan],
        [3, 4, 7],
        [6, 5, np.nan],
        [7, 8, 15],
        [9, 10, 19]
       ]

# Use pandas to handle missing values and convert data to a DataFrame
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
df_dropna = df.dropna()


# Define the target variable
target = df_dropna['C']

# Define the predictor variables
predictors = df_dropna.drop(columns='C')

# Train the linear regression model
reg = LinearRegression().fit(predictors, target)

# Use the trained model to predict values for Target
predictions = reg.predict(predictors)
print("Predictions: ", predictions)
```

```
Predictions:  [ 7. 15. 19.]
```

Column C has missing values. Lets use LR to predict them.

```
df
```

	A	B	C
0	1	2	NaN
1	3	4	7.0
2	6	5	NaN
3	7	8	15.0
4	9	10	19.0

Handling Missing Values with **Linear Regression**

Column C has missing values. Lets use LR to predict them.

```
# Define the predictor variables
predictors = df.drop(columns='C')
# Use the trained model to predict values for missing data
predictions = reg.predict(predictors)
print("Predictions: ", predictions)
```

```
Predictions:  [ 3.  7. 11. 15. 19.]
```

```
df
```

	A	B	C
0	1	2	NaN
1	3	4	7.0
2	6	5	NaN
3	7	8	15.0
4	9	10	19.0

Handling Duplicates

Having duplicate values in a dataset can cause a number of problems when building predictive models or analyzing data. Some of the main problems include:

1. Inaccurate results: Duplicate values can result in inaccuracies when calculating statistics such as mean, median, or mode.
2. Increased storage and computation time: Duplicate values increase the size of the dataset and can slow down the processing time.
3. Overfitting: Duplicate values can increase the risk of overfitting in predictive models, as the model may learn to memorize the duplicates rather than generalize to new data.

Handling Duplicates

```python
import pandas as pd

# Generate synthetic data with duplicate values
data = [[1, 2, 3], [3, 4, 5], [6, 7, 8], [1, 2, 3], [9, 10, 11]]

# Use pandas to convert data to a DataFrame
df = pd.DataFrame(data, columns=['A', 'B', 'C'])

# Calculate mean for column C
mean = df['C'].mean()
print("Mean of column C: ", mean)

# Calculate mean for column C after removing duplicates
df_drop_duplicates = df.drop_duplicates()
mean_without_duplicates = df_drop_duplicates['C'].mean()
print("Mean of column C after removing duplicates: ", mean_without_duplicates)
```

```
Mean of column C:  6.0
Mean of column C after removing duplicates:  6.75
```

```
df
```

	A	B	C
0	1	2	3
1	3	4	5
2	6	7	8
3	1	2	3
4	9	10	11

```
df_drop_duplicates
```

	A	B	C
0	1	2	3
1	3	4	5
2	6	7	8
4	9	10	11

Handling Duplicates

```
import pandas as pd

# Load the data into a Pandas DataFrame
df = pd.read_csv('data.csv')

# Drop duplicate rows
df.drop_duplicates(inplace=True)

# Drop duplicates based on multiple columns
df.drop_duplicates(subset=['col1', 'col2'], inplace=True)
```

You can drop duplicates based on one or multiple columns.

```
import pandas as pd

# Generate synthetic data with duplicate values
data = [[1, 2, 3], [3, 4, 5], [6, 7, 8], [1, 2, 3], [9, 10, 11]]

# Use pandas to convert data to a DataFrame
df = pd.DataFrame(data, columns=['A', 'B', 'C'])

# Calculate mean for column C
mean = df['C'].mean()
print("Mean of column C: ", mean)

# Calculate mean for column C after removing duplicates
df_drop_duplicates = df.drop_duplicates()
mean_without_duplicates = df_drop_duplicates['C'].mean()
print("Mean of column C after removing duplicates: ", mean_without_duplicates)
```

```
Mean of column C:  6.0
Mean of column C after removing duplicates:  6.75
```

Outliers

Outliers in a dataset can cause problems when building predictive models or analyzing data. Some of the main problems include:

1. Inaccurate results: Outliers can skew the results of statistical analyses and impact the accuracy of predictive models.
2. Increased variance: Outliers can increase the variance of the data, which can make it more difficult to build accurate models.
3. Overfitting: Outliers can increase the risk of overfitting in predictive models, as the model may learn to memorize the outliers rather than generalize to new data.

Outliers

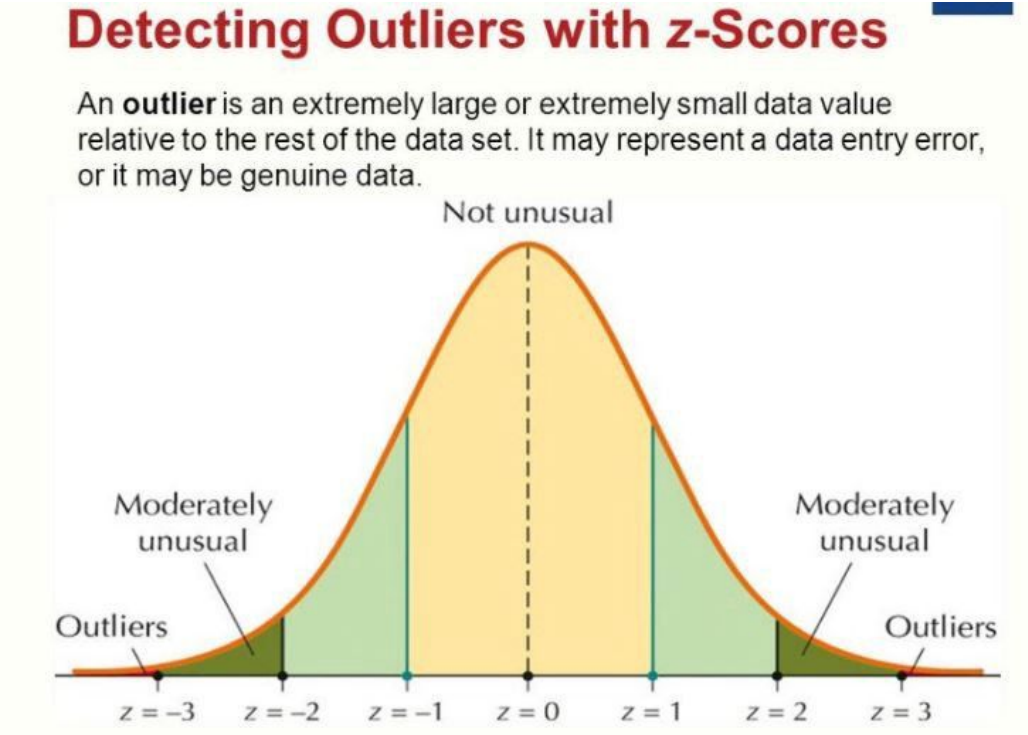
To handle outliers, one common approach is to remove them or to replace them with more representative values. This can be done using various techniques such as:

1. Z-score method
2. Modified z-score Method
3. IQR (Interquartile Range) method

Handling Outliers with **Z-Score (K-sigma)**

$$Z = \frac{x_i - \bar{x}}{s}$$

Here, X-bar is the mean value and s is standard deviation.



Handling Outliers with Z-Score (K-sigma)

```python
import numpy as np
import pandas as pd

# Generate a synthetic data set with 100 samples and 5 features
np.random.seed(0)
data = np.random.randn(12, 5)

cs = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']
# Convert the data to a pandas DataFrame
df = pd.DataFrame(data, columns = cs)

# Introduce one outlier into the DataFrame
df.iloc[-1, :] = [1, 200, 300, 200, 100]
df
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	-1.980796	-0.347912
7	0.156349	1.230291	1.202380	-0.387327	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740
10	-0.895467	0.386902	-0.510805	-1.180632	-0.028182
11	1.000000	200.000000	300.000000	200.000000	100.000000

Handling Outliers with Z-Score (K-sigma)

```
# Calculate the Z-score of each value in the dataset
z = np.abs(df - df.mean()) / df.std()
z
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.797131	0.288509	0.278339	0.250029	0.236889
1	0.569280	0.278967	0.291391	0.290622	0.287523
2	0.398684	0.270218	0.280853	0.286728	0.286367
3	0.562379	0.269527	0.292012	0.283413	0.331476
4	1.929488	0.284111	0.279659	0.301687	0.222911
5	0.981118	0.294659	0.291805	0.262292	0.250728
6	0.408096	0.288891	0.299897	0.323136	0.313884
7	0.409306	0.274105	0.275756	0.295542	0.312299
8	0.630807	0.320094	0.309350	0.255053	0.319505
9	0.103821	0.317192	0.280663	0.316783	0.309187
10	0.498657	0.288739	0.295542	0.309280	0.302772
11	1.137575	3.175012	3.175266	3.174564	3.173543

Handling Outliers with Z-Score (K-sigma)

```
threshold = 3

z < threshold
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	True	True	True	True	True
1	True	True	True	True	True
2	True	True	True	True	True
3	True	True	True	True	True
4	True	True	True	True	True
5	True	True	True	True	True
6	True	True	True	True	True
7	True	True	True	True	True
8	True	True	True	True	True
9	True	True	True	True	True
10	True	True	True	True	True
11	True	False	False	False	False

```
df[(z < threshold)]
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	-1.980796	-0.347912
7	0.156349	1.230291	1.202380	-0.387327	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740
10	-0.895467	0.386902	-0.510805	-1.180632	-0.028182
11	1.000000	NaN	NaN	NaN	NaN

Handling Outliers with Z-Score (K-sigma)

```
# Remove rows with Z-scores above a threshold
df = df[(z < threshold).all(axis=1)]
df
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	-1.980796	-0.347912
7	0.156349	1.230291	1.202380	-0.387327	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740
10	-0.895467	0.386902	-0.510805	-1.180632	-0.028182

```
# Check for outliers
print("Number of rows removed due to outliers: ", len(data) - len(df))
```

```
Number of rows removed due to outliers:  1
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	0.154947	0.378163	-0.887786	-1.980796	-0.347912
7	0.156349	1.230291	1.202380	-0.387327	-0.302303
8	-1.048553	-1.420018	-1.706270	1.950775	-0.509652
9	-0.438074	-1.252795	0.777490	-1.613898	-0.212740
10	-0.895467	0.386902	-0.510805	-1.180632	-0.028182
11	1.000000	200.000000	300.000000	200.000000	100.000000

Modified Z-Score with Mean Absolute Deviation

In statistics, the **median absolute deviation** (**MAD**) is a robust measure of the variability of a univariate sample of quantitative data.

$$y_i = \frac{x_i - \tilde{X}}{\text{MAD}}$$

$$\tilde{X} = \text{median of } X$$

$$\text{MAD} = \text{median}(|\, x_i - \tilde{X} \,|)$$

MAD stands for **m**edian **a**bsolute **d**eviation from the median.

Source: link

Data

```python
import numpy as np
import pandas as pd

# Generate a synthetic data set with 100 samples and 5 features
np.random.seed(0)
data = np.random.randn(7, 5)

cs = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']
# Convert the data to a pandas DataFrame
df = pd.DataFrame(data, columns = cs)

# Introduce one outlier into the DataFrame
df.iloc[-1, :] = [1, 200, 300, 200, 100]
df
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	1.000000	200.000000	300.000000	200.000000	100.000000

Z-Score

```
# Calculate the Z-score of each value in the dataset
z = np.abs(df - df.mean()) / df.std()

threshold = 3
# Remove rows with Z-scores above a threshold
df = df[(z < threshold).all(axis=1)]

# Check for outliers
print("Number of rows removed due to outliers: ", len(data) - len(df))

df
```

```
Number of rows removed due to outliers:  0
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	1.000000	200.000000	300.000000	200.000000	100.000000

Z-score is susceptible to extreme data points. If there is one extreme value, the z-score corresponding to that point will also be extreme.

It has the potential to significantly move the mean away from its actual value.

```
df_Median_Minus = np.abs(df - df.median())

# MAD stands for median absolute deviation from the median.
MAD = df_Median_Minus.median()

# Calculate the Modified Z-score of each value in the dataset
z = np.abs(df - df.median()) / MAD

threshold = 3
# Remove rows with Z-scores above a threshold
df = df[(z < threshold).all(axis=1)]

# Check for outliers
print("Number of rows removed due to outliers: ", len(data) - len(df))

df
```

```
Number of rows removed due to outliers:  1
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359

Modified Z-Score

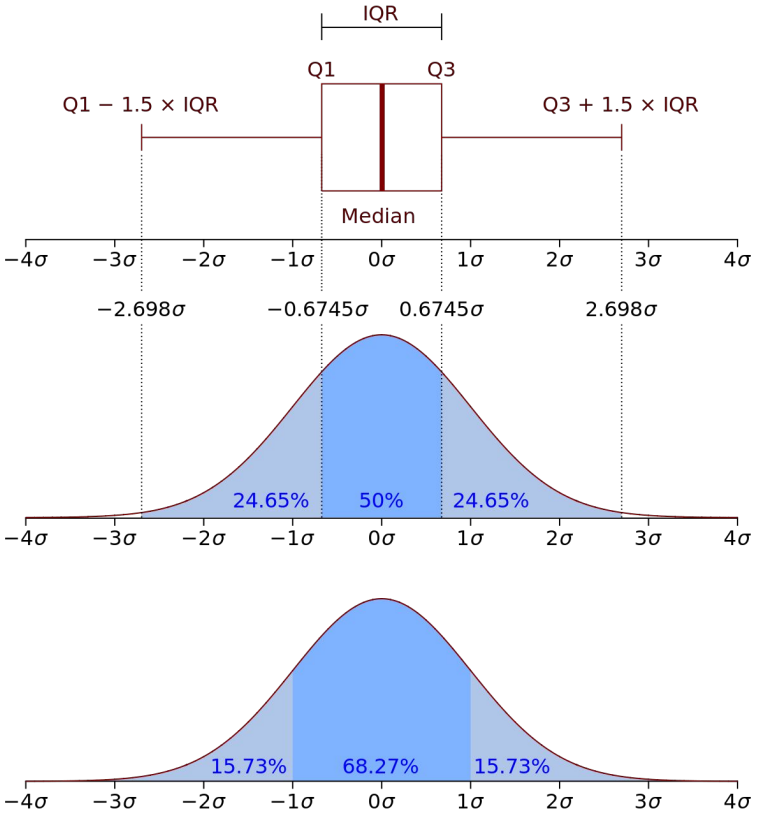
In statistics, the **median absolute deviation** (**MAD**) is a robust measure of the variability of a univariate sample of quantitative data.

Handling Outliers with Inter-Quartile Range

Inter Quartile Range (IQR) is one of the most extensively used procedure for outlier detection and removal.

- Find the first quartile, Q1.
- Find the third quartile, Q3.
- Calculate the IQR. IQR = Q3-Q1.
- Define the normal data range with lower limit as Q1–1.5*IQR and upper limit as Q3+1.5*IQR.
- Any data point outside this range is considered as outlier and should be removed for further analysis.

Source: Link

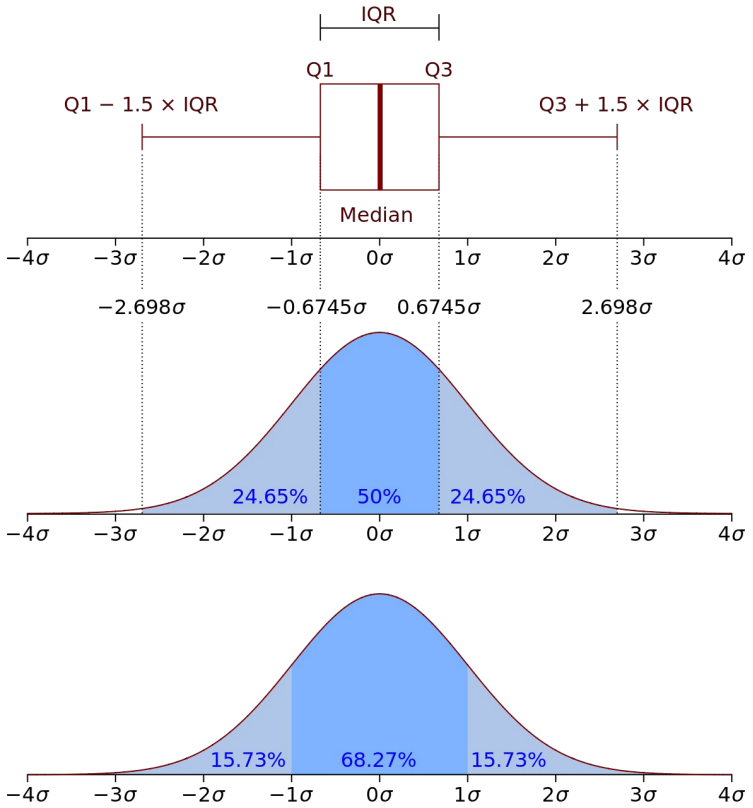


Handling Outliers with Inter-Quartile Range

wikipedia

i	x[i]	Median	Quartile
1	7		
2	7		
3	31		Q_1=31 (median of upper half, from row 1 to 6)
4	31		
5	47		
6	75		
7	87	Q_2=87 (median of whole table)	
8	115		
9	116		
10	119		Q_3=119 (median of lower half, from row 8 to 13)
11	119		
12	155		
13	177		

For the data in this table the interquartile range is IQR = $Q_3 - Q_1$ = 119 - 31 = 88.

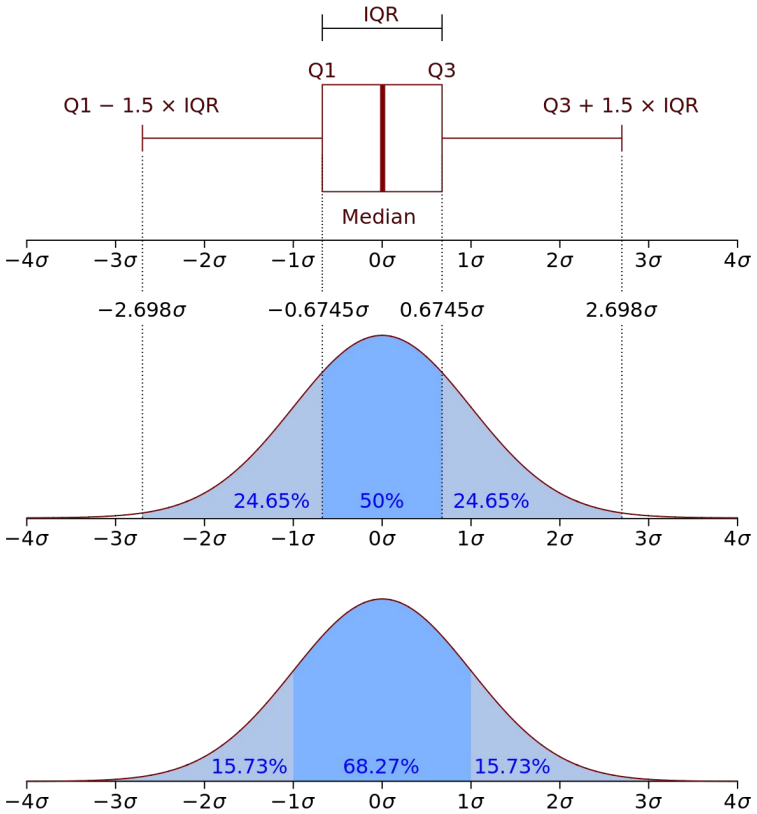


Handling Outliers IQR

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
```

```
outlier = (df < lower_limit)|(df > upper_limit)
outlier
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	True	True	True	True



Handling Outliers IQR

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
```

```
outlier = (df < lower_limit)|(df > upper_limit)
outlier
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	True	True	True	True

```
df[~outlier]
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	1.000000	NaN	NaN	NaN	NaN

```
# The any() method returns one value for each column,
# True if ANY value in that column is True, otherwise False.
df[~outlier.any(axis=1)]
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

# Scale the data using StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

You can scale the data to ensure that each feature has a similar range and scale.

Handling missing and incorrect data

```
import pandas as pd

# Load dataset
df = pd.read_csv("data.csv")

# Remove rows with missing values
df = df.dropna()

# Fill missing values with mean
df = df.fillna(df.mean())
```

It can impact the quality and accuracy of the results obtained from the analysis.

- Removing the rows or columns containing missing data.
- Replacing missing data with mean, median or mode of the column.

Handling Incorrect Data:

```
# Correct invalid values using rules
df.loc[df['column_name'] < 0, 'column_name'] = 0

# Correct invalid values using domain knowledge
df.loc[df['column_name'] == 'Unknown', 'column_name'] = 'New York'
```

Incorrect data can be corrected by validating the data against a set of rules or by using domain knowledge to fill in the correct values.

Data transformation

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv("data.csv")

# Log transformation
df['column_name'] = np.log(df['column_name'])

# Square root transformation
df['column_name'] = np.sqrt(df['column_name'])

# Box-Cox transformation
df['column_name'],_ = stats.boxcox(df['column_name'])
```

Data transformation involves converting the data into a format that can be effectively analyzed and processed. Common data transformations include scaling, log transformation, square root transformation, and box-cox transformation.

Data transformation II

```
import pandas as pd

# Load dataset
df = pd.read_csv("data.csv")

# Aggregation
df = df.groupby(['column_name']).agg({'value': 'sum'})

# Pivot
df = df.pivot(index='column_name', columns='value', values='value')
```

Data transformation is the process of converting data from one format or structure to another.

Common transformations include scaling, aggregating, and pivoting.

Data transformation III: Log Transformation

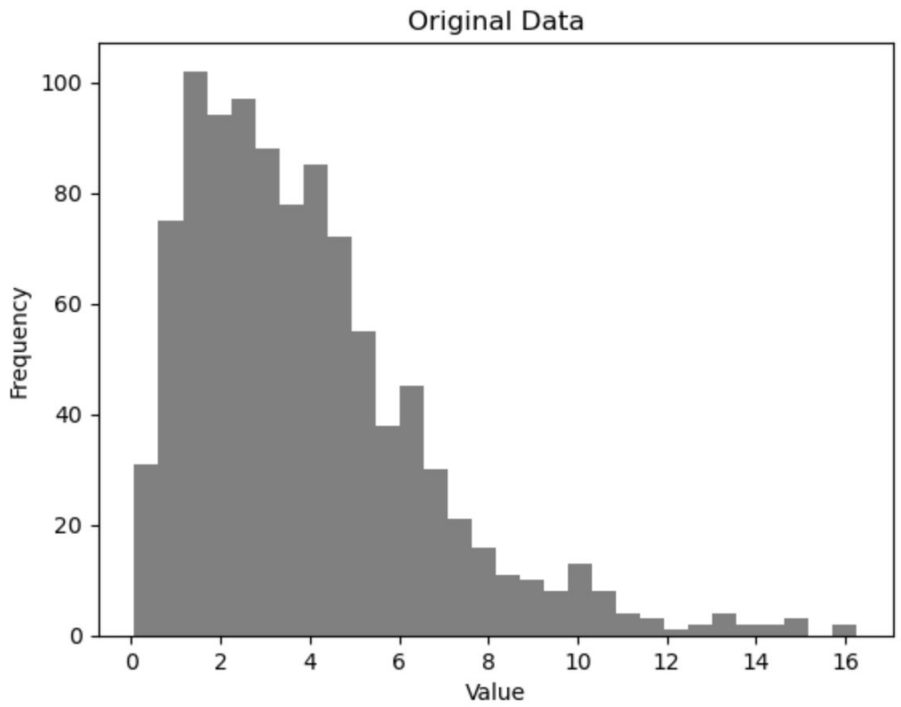
Here are a few scenarios where log transformations can be useful:

1. Right-Skewed Distribution: If a dataset has a right-skewed distribution, meaning the majority of the values are concentrated on the left side of the distribution with a long tail to the right, a log transformation can help to reduce the skew and make the distribution more normal.
2. Highly Skewed Data: If a dataset has extreme outliers or a highly skewed distribution, a log transformation can help to reduce the impact of these outliers and make the distribution more symmetrical.

Log transformations are often used in data science projects to transform skewed or highly skewed datasets into a more normal or Gaussian-like distribution.

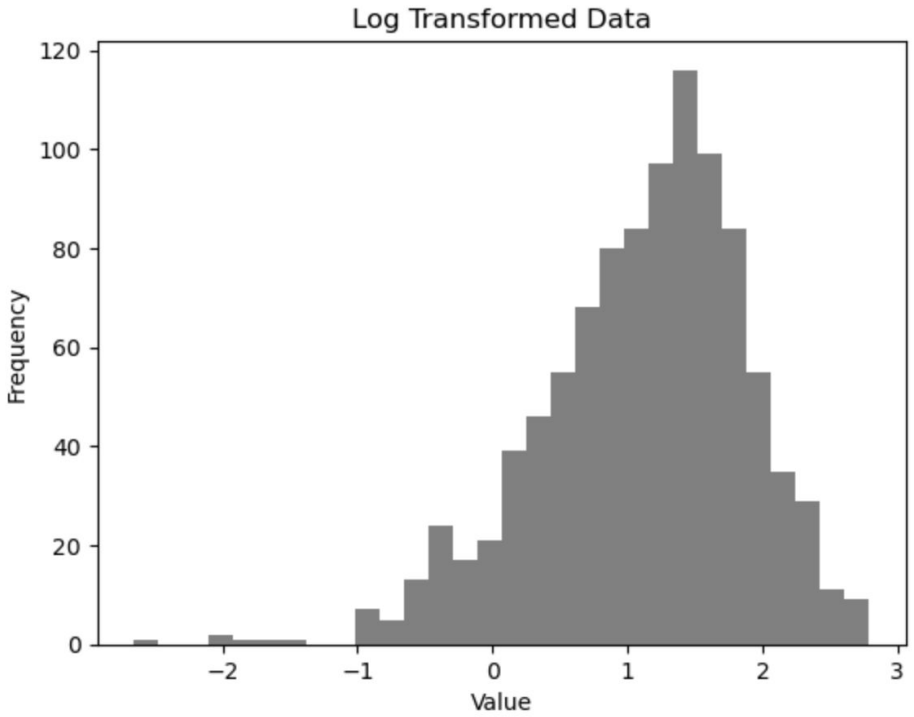
Data transformation III: Log Transformation

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate skewed data using numpy's random.gamma function
np.random.seed(0)
data = np.random.gamma(2, 2, 1000)

# Plot the original data
plt.hist(data, bins=30, color='gray')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Original Data')
plt.show()

# Apply log transformation to the data
log_data = np.log(data)

# Plot the transformed data
plt.hist(log_data, bins=30, color='gray')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Log Transformed Data')
plt.show()
```



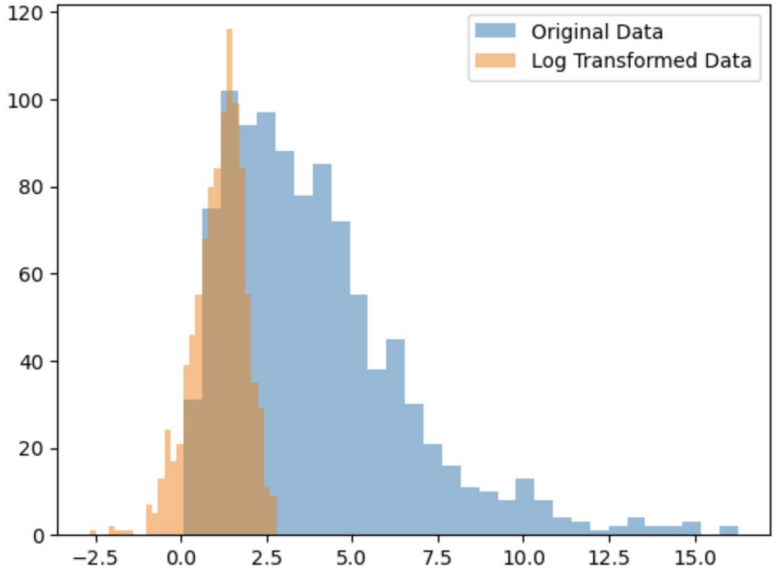
Data transformation III: Log Transformation

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate skewed data using numpy's random.gamma function
np.random.seed(0)
data = np.random.gamma(2, 2, 1000)

# Plot the original data
plt.hist(data, bins=30, color='gray')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Original Data')
plt.show()

# Apply log transformation to the data
log_data = np.log(data)

# Plot the transformed data
plt.hist(log_data, bins=30, color='gray')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Log Transformed Data')
plt.show()
```



Data transformation III: Log Transformation

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Generate a skewed distribution with positive values
np.random.seed(0)
data = np.random.gamma(2, 2, 1000)

# Plot the original data
plt.hist(data, bins=30, alpha=0.5, label='Original Data')

# Apply log transformation to the data
log_data = np.log(data)

# Plot the log transformed data
plt.hist(log_data, bins=30, alpha=0.5, label='Log Transformed Data')

plt.legend()
plt.show()
```



Log-transformation Plus z-score

```python
import numpy as np
import pandas as pd

# Generate a synthetic data set with 100 samples and 5 features
np.random.seed(0)
data = np.random.randn(7, 5)

cs = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']
# Convert the data to a pandas DataFrame
df = pd.DataFrame(data, columns = cs)

# Introduce one outlier into the DataFrame
df.iloc[-1, :] = [1, 200, 300, 200, 100]
df
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
5	-1.454366	0.045759	-0.187184	1.532779	1.469359
6	1.000000	200.000000	300.000000	200.000000	100.000000

```python
# Apply log transformation to the data
log_df = np.log(df + abs(min(df.min())) + 1)

# Calculate the Z-score of each value in the dataset
z = np.abs(log_df - log_df.mean()) / log_df.std()

threshold = 2
# Remove rows with Z-scores above a threshold
log_df = log_df[(z < threshold).all(axis=1)]

# Check for outliers
print("Number of rows removed due to outliers: ", len(data) - len(log_df))

log_df
```

```
Number of rows removed due to outliers:  1
```

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0	1.670917	1.374512	1.511103	1.756803	1.690197
1	0.946126	1.504761	1.224255	1.238308	1.377150
2	1.307531	1.610890	1.461872	1.301462	1.385507
3	1.357551	1.618808	1.208313	1.352235	0.992842
4	0.000000	1.436657	1.485557	1.033478	1.761772
5	0.741282	1.280586	1.213667	1.626446	1.613898

Data Normalization MIN-MAX

```
# Min-Max normalization
df['column_name']
      = (df['column_name'] - df['column_name'].min())
      / (df['column_name'].max() -df['column_name'].min())
```

Data normalization involves scaling the data to a common range, such as 0 to 1, to remove any bias and help the algorithms to converge faster.

Common data normalization techniques include Min-Max normalization and Z-Score normalization.

Data Normalization Standard Scaler

```
# Z-Score normalization
df['column_name']
      = (df['column_name'] -df['column_name'].mean())
      / df['column_name'].std()
```

Data normalization is the process of scaling the data so that it has a mean of zero and a standard deviation of one.

This is important for algorithms that are sensitive to the scale of the data, such as k-nearest neighbors and neural networks.

Data Normalization Standard Scaler II

```
from sklearn.preprocessing import StandardScaler

# Scale the data using StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

You can scale the data to ensure that each feature has a similar range and scale.

How to deal with Class Imbalance in Python

1. Resampling: You can oversample the minority class or undersample the majority class to balance the class distribution.
2. Synthetic Data Generation: Using techniques such as Synthetic Minority Over-sampling Technique (SMOTE) to generate new synthetic samples for the minority class.
3. Cost-sensitive learning: You can assign different misclassification costs to different classes based on their imbalanced ratio, and adjust the model's decision boundary to take these costs into account.
4. Using different evaluation metrics: Instead of relying on accuracy, use metrics like F1-score, AUC-ROC, Precision-Recall, etc., which are more sensitive to class imbalance.

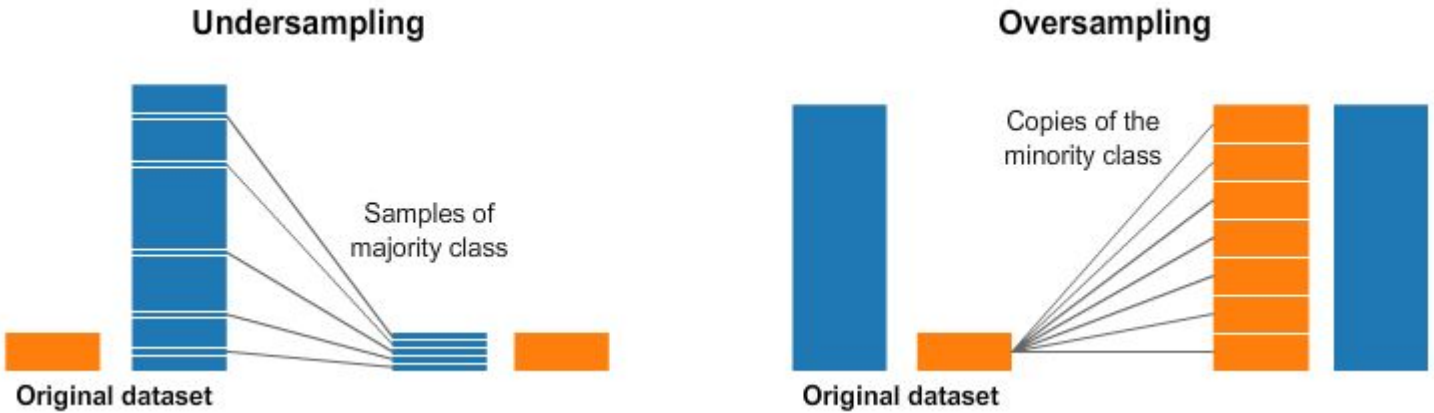
In Python, you can use libraries such as imbalanced-learn, scikit-learn, and xgboost to handle class imbalance.

How to deal with Class Imbalance in Python

The Metric Trap

The classifier will always “predicts” the most common class without performing any analysis of the features and it will have a high accuracy rate, obviously not the correct one.

Resampling



Source: link

Random Undersampling

Class Imbalance

```
from sklearn.datasets import make_classification

X, y = make_classification(n_classes=2,
                           weights=[0.9, 0.1],
                           n_features=5,
                           n_samples=100,
                           random_state=10)
```

```
X.shape
```

```
(100, 5)
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

```
data = pd.DataFrame(X)
data['Class'] = y
data
```

	0	1	2	3	4	Class
0	-1.459734	-1.196858	0.982202	0.540541	-1.261557	0
1	-1.751322	-1.579690	1.094644	-2.017719	-0.900804	0
2	-1.933815	-1.543645	1.325618	0.148961	-1.849956	0
3	0.176859	0.481785	0.077217	0.892564	-1.282663	0
4	-1.263675	-0.987174	0.878791	0.279605	-1.300690	0
...	...	...	...	...	...	...
95	-1.338862	-1.219320	0.830043	1.674076	-0.638919	0
96	-1.883909	-1.814648	1.110300	0.547944	-0.477258	0
97	1.443844	1.795983	-0.614846	-0.857523	-1.361483	0
98	-1.131836	-0.491450	1.015928	1.412064	-2.839013	0
99	0.246815	0.429348	-0.033826	0.264395	-0.754197	0

100 rows × 6 columns

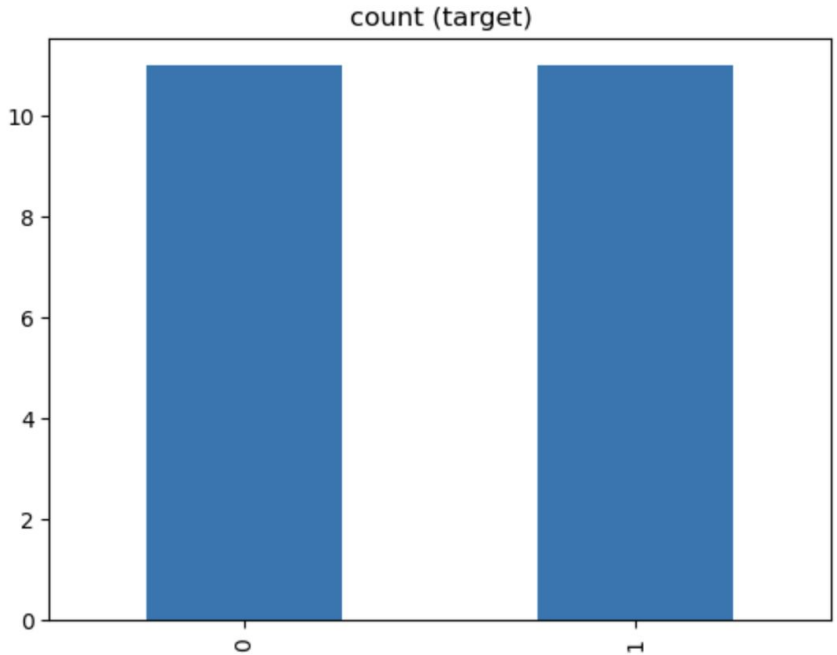
Random Undersampling

```
# class count
class_count_0, class_count_1 = data['Class'].value_counts()

# Separate class
class_0 = data[data['Class'] == 0]
class_1 = data[data['Class'] == 1]

# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (89, 6)
class 1: (11, 6)
```

```
class_0_under = class_0.sample(class_count_1)
test_under = pd.concat([class_0_under, class_1], axis=0)

print("total class of 1 and 0: \n",test_under['Class'].value_counts())

# plot the count after under-sampeling
test_under['Class'].value_counts().plot(kind='bar', title='count (target)')
```

```
total class of 1 and 0:
 0    11
1    11
Name: Class, dtype: int64

<AxesSubplot: title={'center': 'count (target)'}>
```



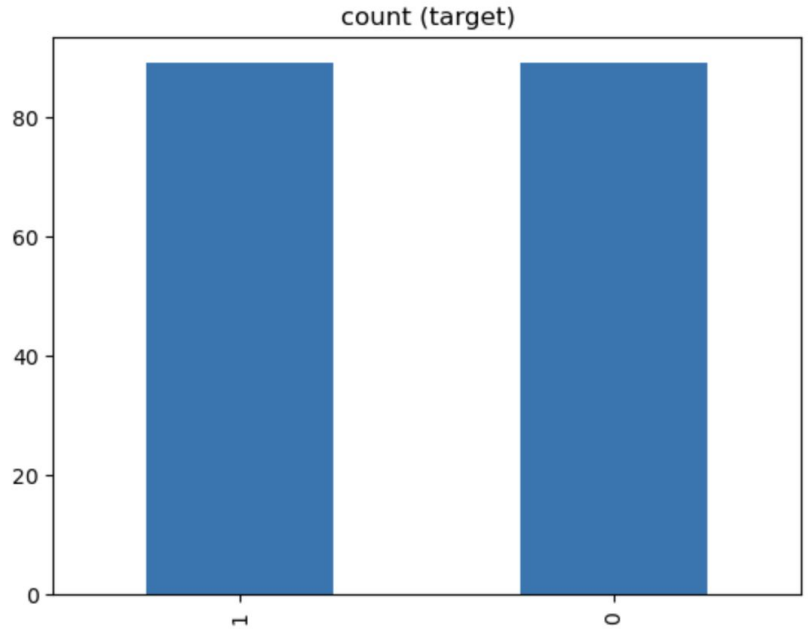
Random Oversampling

```
# class count
class_count_0, class_count_1 = data['Class'].value_counts()

# Separate class
class_0 = data[data['Class'] == 0]
class_1 = data[data['Class'] == 1]

# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (89, 6)
class 1: (11, 6)
```

```
# Random Over-Sampling
class_1_over = class_1.sample(class_count_0, replace=True)

test_over = pd.concat([class_1_over, class_0], axis=0)

print("total class of 1 and 0: \n",test_under['Class'].value_counts())

# plot the count after under-sampeling
test_over['Class'].value_counts().plot(kind='bar', title='count (target)')
```

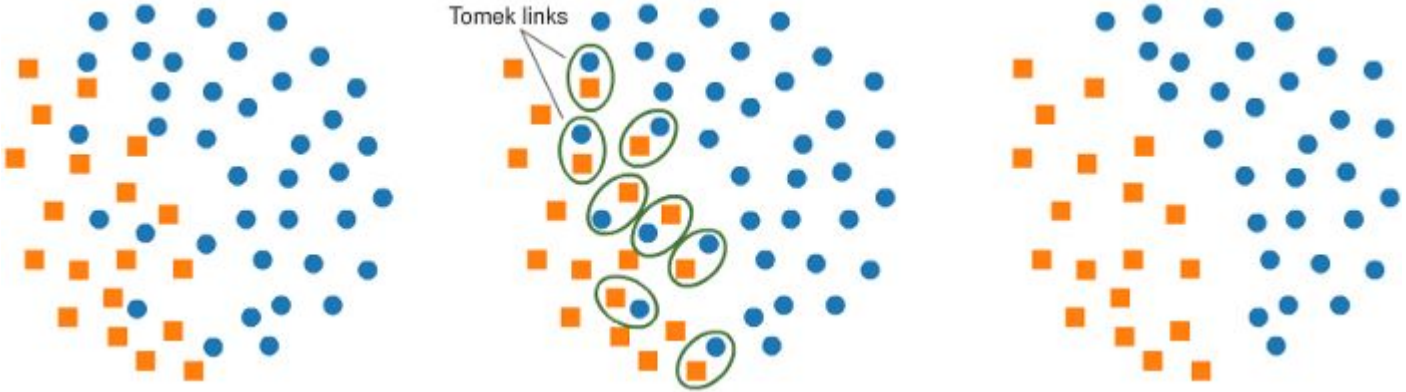
```
total class of 1 and 0:
 0    11
1    11
Name: Class, dtype: int64

<AxesSubplot: title={'center': 'count (target)'}>
```



How to deal with Class Imbalance in Python

Under-sampling: Tomek links

Tomek links are pairs of very close instances but of opposite classes. Removing the instances of the majority class of each pair increases the space between the two classes, facilitating the classification process. Tomek's link exists if the two samples are the nearest neighbors of each other

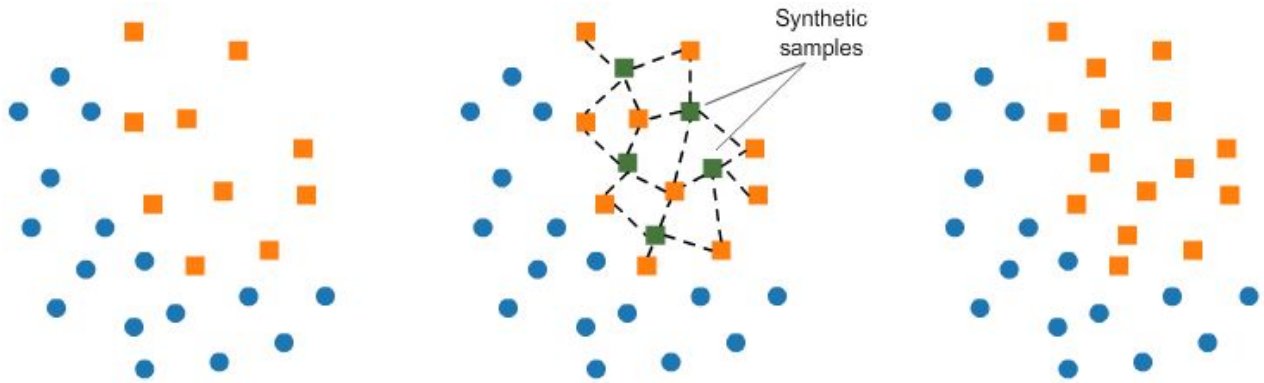


Source: link

How to deal with Class Imbalance in Python

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.



Source: link