# Lecture_material_3

February 24, 2024

## 1 Data wrangling

Data wrangling, also known as data munging, is the process of cleaning, structuring, and organizing raw data into a format suitable for analysis.

It involves transforming and mapping data from its raw form into another format to make it more appropriate and valuable for a variety of downstream purposes, such as analysis or visualization.

## 2 Data wrangling (Key Activities)

Handling missing or inconsistent data.
Transforming variables or features.
Merging or joining datasets.
Reshaping data (e.g., pivoting, melting).

## 3 Data wrangling (steps)

Importing necessary libraries such as Pandas, NumPy, and Matplotlib.
Loading the data into a Pandas DataFrame.
Assessing the data for missing values, outliers, and inconsistencies.
Cleaning the data by filling in missing values, removing outliers, and correcting errors.
Organizing the data by creating new columns, renaming columns, sorting, and filtering the data.
Storing the cleaned data in a format that can be used for future analysis, such as a CSV or Excel file.
Exploring the data by creating visualizations and using descriptive statistics.
Creating a pivot table to summarize the data.
Checking for and handling duplicate rows.
Encoding categorical variables.
Removing unnecessary columns or rows.
Merging or joining multiple datasets
Handling missing or null values
Reshaping the data
Formatting the data
Normalizing or scaling the data
Creating new features from existing data
Validating data integrity
Saving the final data for future use
Documenting the data wrangling process for reproducibility

## 4 Data mining

is the process of discovering patterns, relationships, and information from large amounts of data. It involves extracting valuable insights or knowledge from data using various techniques, including statistical analysis, machine learning, and pattern recognition. # Key Activities Exploratory Data Analysis (EDA).
Feature extraction and selection.
Pattern recognition and clustering.
Predictive modeling and classification.

## 5 Data Cleaning

Data cleaning is the process of identifying and correcting (or removing) errors and inconsistencies in datasets. The goal is to improve the quality of the data by addressing issues such as missing values, duplicates, outliers, and inaccuracies. # Key Activities: Handling missing data (imputation or removal).
Identifying and removing duplicate records.
Detecting and addressing outliers.
Standardizing and validating data formats.
Correcting errors and inconsistencies.

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
df=sns.load_dataset('titanic')
df.rename(columns={"sex": "gender"}, inplace=True)
```

[ ]:
```
df.drop(['deck'],axis=1,inplace=True)
```

[ ]:
```
df.head()
```

[ ]:
```
#use loc function to select rows and columns by label
df.loc[0:4,'age':"fare"]
```

[ ]:
```
# use iloc function to select rows and columns by position
df.iloc[0:4,0:4]
```

[ ]:
```
# if we wanti to excess from 0 to 4 rows then 8 to 12 rows and 0 to 4 columns␣
 ↪using loc function
df.loc[np.r_[0:4,8:12],['age','fare']]
```

[ ]:
```
# if we wanti to excess from 0 to 4 rows then 8 to 12 rows and 0 to 4 columns
df.iloc[np.r_[0:4,8:12],0:4]
```

[ ]:
```
sns.boxplot(df, x='gender', y="age")
```

```
[ ]: # Inter quartile range method
     Q1=df['age'].quantile(0.25)
     Q3=df['age'].quantile(0.75)
     IQR=Q3-Q1
     lower_limit=Q1-1.5*IQR
     upper_limit=Q3+1.5*IQR
```

```
[ ]: df=df[(df['age']>lower_limit) & (df['age']<upper_limit)]
```

```
[ ]: Q1=df['fare'].quantile(0.25)
     Q3=df['fare'].quantile(0.75)
     IQR=Q3-Q1
     lower_limit=Q1-1.5*IQR
     upper_limit=Q3+1.5*IQR
     df=df[(df['fare']>lower_limit) & (df['fare']<upper_limit)]
```

```
[ ]: df.shape
```

```
[ ]: from scipy import stats
```

```
[ ]: df_duplicates=df[df.duplicated()]
     df_duplicates.shape
```

```
[ ]: df.drop_duplicates(inplace=True)
```

```
[ ]: df.shape
```

```
[ ]: from sklearn.preprocessing import MinMaxScaler
     scaler=MinMaxScaler()
     df[['age','fare']]=scaler.fit_transform(df[['age','fare']])
     df
```

## 6 Organizing the data

```
[ ]: df["family"]=df['sibsp']+df['parch']
```

```
[ ]: import seaborn as sns
     sns.swarmplot(data=df, x="gender", y="family")
```

```
[ ]: sns.swarmplot(data=df, x="gender", y="age",hue='family')
```

```
[ ]: table=pd.pivot_table(df, values="fare", index="pclass", columns="survived",␣
      ↪aggfunc=np.sum)
     table
```

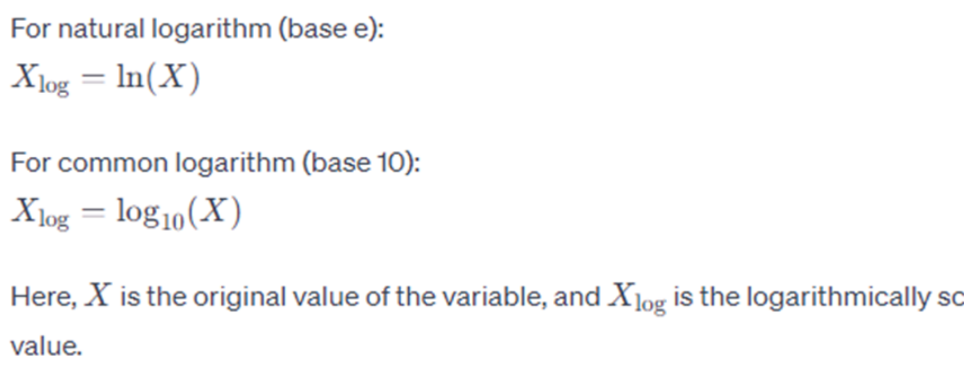
## 7 Feature Scaling

Feature scaling is a preprocessing technique used in machine learning to standardize or normalize the range of independent variables or features of a dataset.

The goal is to bring all features to a similar scale to prevent some features from dominating or having undue influence on the learning algorithm.

Feature scaling is particularly important for algorithms that are sensitive to the scale of input features, such as gradient-based optimization algorithms in machine learning.

### 7.1 1. Min Max Scaling



For natural logarithm (base e):

$$X_{\log} = \ln(X)$$

For common logarithm (base 10):

$$X_{\log} = \log_{10}(X)$$

Here, $X$ is the original value of the variable, and $X_{\log}$ is the logarithmically sc

value.

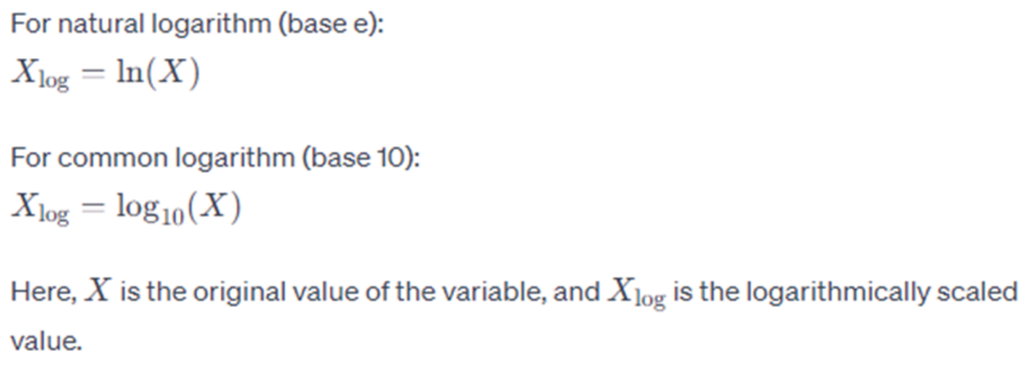
Scales the values between 0 and 1.

```
[ ]: import pandas as pd
     from sklearn.preprocessing import MinMaxScaler
     # Sample data
     data = {'numbers': [10, 20, 30, 40, 50]}
     df = pd.DataFrame(data)
     df.head()
```

```
[ ]: # scale the data using min max scalar
     scaler = MinMaxScaler()
     df['numbers_scaled'] = scaler.fit_transform(df[['numbers']])
     df.head()
```

## 8 2. Standard Scalar or Z-score normalization

Scales the values to have a mean of 0 and a standard deviation of 1



For natural logarithm (base e):

$$X_{\log} = \ln(X)$$

For common logarithm (base 10):

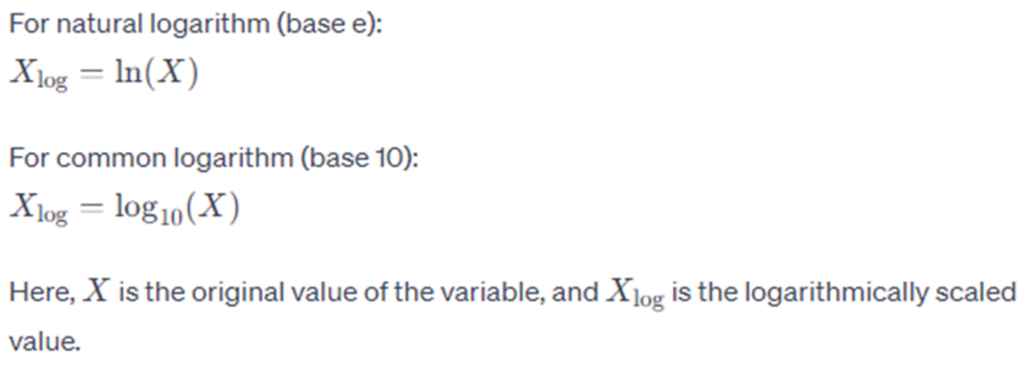
$$X_{\log} = \log_{10}(X)$$

Here, $X$ is the original value of the variable, and $X_{\log}$ is the logarithmically scaled value.

```
[ ]: # scale the data using standard scalar
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     df['numbers_scaled'] = scaler.fit_transform(df[['numbers']])
     df.head()
```

## 9 3. Robust scalar

Scales the data based on the interquartile range (IQR).



For natural logarithm (base e):

$$X_{\log} = \ln(X)$$

For common logarithm (base 10):

$$X_{\log} = \log_{10}(X)$$

Here, $X$ is the original value of the variable, and $X_{\log}$ is the logarithmically scaled value.

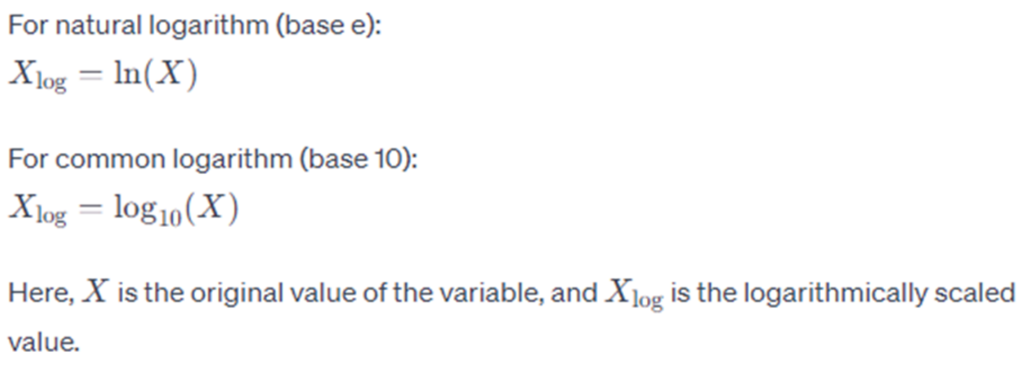
Robust to outliers.

```
[ ]: from sklearn.preprocessing import RobustScaler
     # Sample data with outliers

     # Robust Scaling
     scaler = RobustScaler()
     df['numbers_scaled'] = scaler.fit_transform(df[['numbers']])
     print(df)
```

## 10 4. Logrithmic scaling/ Normalization

Logarithmic scaling, also known as logarithmic normalization, is a type of feature scaling that involves applying a logarithmic transformation to the values of a variable.
The logarithmic function used is typically the natural logarithm (base e) or the common logarithm



For natural logarithm (base e):

$$X_{\log} = \ln(X)$$

For common logarithm (base 10):

$$X_{\log} = \log_{10}(X)$$

Here, $X$ is the original value of the variable, and $X_{\log}$ is the logarithmically scaled value.

(base 10).

[ ]:
```
import numpy as np
import pandas as pd

#random data with outliers

# Log Transform
df['numbers_log'] = np.log(df['numbers'])
df['numbers_log2'] = np.log2(df['numbers'])
df['numbers_log10'] = np.log10(df['numbers'])
df.head()
```

## 11 Feature Encoding

Feature encoding, also known as feature representation, is the process of transforming categorical or non-numeric data into a numerical format that can be used as input for machine learning algorithms.

Many machine learning models require numerical input, and feature encoding is essential when dealing with categorical variables or non-numeric data types

## 12 1. One hot encoding

Represents each category as a binary vector. For each category, one binary variable is set to 1, and the others are set to 0.

[ ]:
```
import pandas as pd
# Sample data
data = {'Color': ['Red', 'Green', 'Blue', 'Red']}
df = pd.DataFrame(data)
print(df)
# One-Hot Encoding
```

```
encoded_data = pd.get_dummies(df, columns=['Color'])
print(encoded_data)
```

## 13 2. Label Encoding

Assigns a unique integer to each category. This is suitable when there is an inherent ordinal relationship among categories.

```
[ ]: from sklearn.preprocessing import LabelEncoder
     # Sample data
     data = {'Animal': ['Dog', 'Cat', 'Bird', 'Dog', "lion"]}
     df = pd.DataFrame(data)
     print(df)

     # Label Encoding
     label_encoder = LabelEncoder()
     df['Animal_encoded'] = label_encoder.fit_transform(df['Animal'])
     print(df)
```

## 14 3.Ordinal Encoding

Assigns integers to categories based on their ordinal relationship. This is suitable when the categories have a meaningful order.

```
[ ]: from sklearn.preprocessing import OrdinalEncoder
     # Sample data
     data = {'Size': ['Small', 'Medium', 'Large', 'Medium']}
     df = pd.DataFrame(data)
     print(df)

     # Ordinal Encoding
     ordinal_encoder = OrdinalEncoder(categories=[['Small', 'Medium', 'Large']])
     df['Size_encoded'] = ordinal_encoder.fit_transform(df[['Size']])
     print(df)
```