**Data Wrangling**

# **Data Wrangling**

## **Data Pre-Processing in Phython**

1. **Data Pre-processing**:
   * Essential step in data analysis.
   * Involves converting raw data into a format suitable for analysis.
   * Also known as data cleaning or data wrangling.
2. **Handling Missing Values**:
   * Missing values occur when data entries are left empty.
   * Techniques to identify and manage these missing values will be discussed.
3. **Data Formats**:
   * Data from different sources may vary in format, units, or conventions.
   * Methods in Python Pandas will be introduced to standardize these values.
4. **Data Normalization**:
   * Different numerical data columns may have varying ranges.
   * Normalization helps bring all data into a similar range for meaningful comparison.
   * Focus on techniques like centering and scaling.
5. **Data Binning**:
   * Binning creates larger categories from numerical values.
   * Useful for comparing groups of data.
6. **Categorical Variables**:
   * Converting categorical values into numeric variables for easier statistical modeling.
7. **Data Manipulation in Python**:
   * Operations are typically performed along columns in data frames.
   * Example: Adding a value to each entry in a column.

## **Dealing with Missing Values in Python**

1. **Definition of Missing Values**:
   * A **missing value** occurs when no data value is stored for a feature in a particular observation. It can appear as:
     + Question mark (?)
     + N/A
     + Zero (0)
     + Blank cell
     + In Python, it is often represented as NaN (Not a Number).
2. **Strategies to Handle Missing Values**:
   * **Check for Actual Values**: If possible, contact the data collector to find the actual value.
   * **Remove Data**: You can drop the entire variable or just the specific entry with the missing value.
   * **Replace Data**: Substitute missing values with estimates, such as:
     + **Mean**: For numerical data, replace with the average.
     + **Mode**: For categorical data, replace with the most common value.
   * **Leave as Missing**: Sometimes, it may be useful to keep the observation even if some features are missing.

Code Examples

1. Dropping Missing Values

Using the **Pandas** library in Python, you can drop rows or columns with missing values using the dropna() method.

import pandas as pd

# Sample DataFrame

data = {

'Price': [20000, None, 15000, 30000],

'Model': ['A', 'B', 'C', None]

}

df = pd.DataFrame(data)

# Dropping rows with any missing values

df\_dropped = df.dropna(axis=0, inplace=True) # axis=0 for rows

print(df)

**Explanation**:

* dropna(axis=0) removes any row that contains at least one missing value. Setting inplace=True modifies the original DataFrame.

2. Replacing Missing Values

You can replace missing values using the fillna() method or replace() method.

# Replacing missing values with the mean for the 'Price' column

mean\_price = df['Price'].mean()

df['Price'].fillna(mean\_price, inplace=True)

# Replacing missing values in 'Model' with the mode

mode\_model = df['Model'].mode()[0]

df['Model'].fillna(mode\_model, inplace=True)

print(df)

**Explanation**:

* fillna(mean\_price) replaces missing values in the 'Price' column with the calculated mean.
* mode()[0] retrieves the most common value in the 'Model' column, which is then used to fill missing values.

Summary

* **Missing values** can significantly impact data analysis.
* Strategies include **removing**, **replacing**, or **keeping** missing data.
* Use **Pandas** methods like dropna() and fillna() to handle missing values effectively.

## **Data Formatting in Python**

1. **Data Formatting**:
   * Data formatting involves bringing data into a common standard to make meaningful comparisons.
   * Example: Different representations of New York City (N.Y., Ny, NY, New York) can complicate analysis.
2. **Dataset Cleaning**:
   * Ensures data is consistent and understandable.
   * Sometimes, unclean data can be useful for specific analyses, like spotting fraud.
3. **Unit Conversion**:
   * Example: Converting miles per gallon to liters per 100 kilometers.
   * Formula: To convert miles per gallon (mpg) to liters per 100 kilometers (L/100km), use: [ \text{L/100km} = \frac{235}{\text{mpg}} ]

**Code Example**:

import pandas as pd

# Sample DataFrame

df = pd.DataFrame({'city-miles per gallon': [25, 30, 20]})

# Convert mpg to L/100km

df['city-liters per 100 kilometers'] = 235 / df['city-miles per gallon']

print(df)

1. **Data Types in Pandas**:
   * Data types must be correctly established for accurate analysis.
   * Common data types include:
     + **Object**: Strings or mixed types
     + **Int64**: Integer values
     + **Float**: Real numbers
2. **Identifying Data Types**:
   * Use dataframe.dtypes to check the data types of each variable.

**Code Example**:

# Check data types

print(df.dtypes)

1. **Converting Data Types**:
   * If a data type is incorrect, use dataframe.astype() to convert it.

**Code Example**:

# Sample DataFrame with incorrect data type

df\_price = pd.DataFrame({'price': ['100', '200', '300']})

# Convert 'price' from object to integer

df\_price['price'] = df\_price['price'].astype(int)

print(df\_price.dtypes)

Summary

* **Data formatting** is crucial for analysis, ensuring consistency and comparability.
* **Unit conversions** can be easily performed using simple formulas in Pandas.
* **Data types** must be checked and corrected to avoid issues in analysis.

## **Data Normalization in Python**

1. **Purpose of Normalization**:
   * Normalization is used to make the ranges of different features consistent.
   * It helps in making statistical analyses easier and ensures fair comparisons between features.
2. **Example of Feature Ranges**:
   * In a dataset, if one feature (e.g., length) ranges from 150 to 250 and another feature (e.g., width) ranges from 50 to 100, normalization helps align these ranges.
3. **Impact of Different Ranges**:
   * If features have vastly different ranges (e.g., age from 0 to 100 and income from 0 to 500,000), the feature with the larger range (income) will disproportionately influence models like linear regression.
4. **Normalization Techniques**:
   * **Simple Feature Scaling**: Divides each value by the maximum value of that feature.
   * **Min-Max Normalization**: Transforms values to a range of 0 to 1 using the formula: [ x' = \frac{x - \text{min}}{\text{max} - \text{min}} ]
   * **Z-Score Normalization**: Standardizes values based on the mean and standard deviation: [ z = \frac{x - \mu}{\sigma} ] where ( \mu ) is the mean and ( \sigma ) is the standard deviation.

Code Examples:

Here are code snippets for each normalization method using Python and the Pandas library.

1. Simple Feature Scaling

import pandas as pd

# Sample data

data = {'length': [150, 200, 250]}

df = pd.DataFrame(data)

# Simple Feature Scaling

df['length\_normalized'] = df['length'] / df['length'].max()

print(df)

**Explanation**: This code divides each value in the 'length' column by the maximum value in that column, resulting in values between 0 and 1.

2. Min-Max Normalization

# Min-Max Normalization

df['length\_min\_max'] = (df['length'] - df['length'].min()) / (df['length'].max() - df['length'].min())

print(df)

**Explanation**: This code subtracts the minimum value from each value in the 'length' column and then divides by the range (max - min), normalizing the values to a range of 0 to 1.

3. Z-Score Normalization

# Z-Score Normalization

df['length\_z\_score'] = (df['length'] - df['length'].mean()) / df['length'].std()

print(df)

**Explanation**: This code calculates the Z-score for each value by subtracting the mean and dividing by the standard deviation, resulting in values that typically range between -3 and 3.

Summary

* **Normalization** is crucial for ensuring that features contribute equally to analyses and models.
* The three methods outlined above provide different ways to achieve this, depending on the specific needs of your analysis.

## **Binning in Python**

* **Binning Definition**: Binning is the process of grouping a set of numerical values into smaller, discrete intervals called bins. This helps in simplifying the data and improving the accuracy of predictive models.
* **Example of Binning**:
  + **Age Binning**: Grouping ages into bins like 0-5, 6-10, 11-15, etc.
  + **Price Binning**: For a price attribute ranging from 5,000 to 45,500, you can categorize it into three bins: low price, medium price, and high price.

Implementation in Python

To implement binning in Python, you can use libraries like **NumPy** and **Pandas**. Here’s a step-by-step explanation along with code:

1. **Import Libraries**:
2. import numpy as np

import pandas as pd

1. **Create Sample Data**: Let's create a sample dataset for car prices.
2. # Sample car prices
3. prices = [5188, 15000, 25000, 30000, 45000, 20000, 35000, 40000, 45000]

df = pd.DataFrame(prices, columns=['Price'])

1. **Define Bins**: You can define the bins using numpy.linspace to create equally spaced intervals.
2. # Define bins

bins = np.linspace(5000, 45000, 4) # 4 equally spaced bins

1. **Create Bin Labels**: Define labels for the bins.
2. # Define bin labels

bin\_labels = ['Low Price', 'Medium Price', 'High Price']

1. **Apply Binning**: Use the pd.cut() function to segment the data into bins.
2. # Apply binning

df['Price Category'] = pd.cut(df['Price'], bins=bins, labels=bin\_labels, include\_lowest=True)

1. **View the Result**: Check the resulting DataFrame.

print(df)

Explanation of the Code

* **Importing Libraries**: We import NumPy for numerical operations and Pandas for data manipulation.
* **Creating Sample Data**: We create a DataFrame with car prices.
* **Defining Bins**: We use np.linspace to create 4 equally spaced bins between 5,000 and 45,000.
* **Creating Bin Labels**: We define labels for each bin to make the output more understandable.
* **Applying Binning**: The pd.cut() function segments the prices into the defined bins and assigns the corresponding labels.
* **Viewing the Result**: Finally, we print the DataFrame to see the categorized prices.

Visualization

You can visualize the distribution of the data using a histogram:

import matplotlib.pyplot as plt

# Plot histogram

df['Price'].hist(bins=bins, edgecolor='black')

plt.title('Price Distribution')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.show()

This histogram will help you see how many cars fall into each price category.

**Turning Categorical variables into Quantitative variables in Python** turning categorical variables into quantitative variables in Python, along with relevant code examples and explanations.

Key Points:

1. **Categorical Variables**: These are variables that represent categories or groups, such as "fuel type" in a car dataset, which can have values like "gas" or "diesel".
2. **Need for Conversion**: Most statistical models require numerical input, so categorical variables must be converted into a numeric format for analysis.
3. **One Hot Encoding**:
   * This technique creates new binary (0 or 1) features for each unique category in the original variable.
   * For example, if the "fuel type" feature has two categories (gas and diesel), two new features will be created: "gas" and "diesel".
   * If a car uses diesel, the "diesel" feature will be set to 1, and the "gas" feature will be set to 0, and vice versa.

Example Code:

Here’s how you can implement one hot encoding in Python using the pandas library:

import pandas as pd

# Sample data

data = {

'car': ['A', 'B', 'C', 'D'],

'fuel\_type': ['gas', 'diesel', 'diesel', 'gas']

}

# Create a DataFrame

df = pd.DataFrame(data)

# Apply one hot encoding

dummy\_variable\_one = pd.get\_dummies(df['fuel\_type'], prefix='fuel')

# Concatenate the original DataFrame with the new dummy variables

df = pd.concat([df, dummy\_variable\_one], axis=1)

# Display the updated DataFrame

print(df)

Explanation of the Code:

* **Importing pandas**: The pandas library is imported to handle data manipulation.
* **Creating Sample Data**: A dictionary is created with car names and their corresponding fuel types.
* **Creating a DataFrame**: The dictionary is converted into a DataFrame for easier manipulation.
* **One Hot Encoding**: The pd.get\_dummies() function is used to convert the "fuel\_type" column into dummy variables. The prefix argument adds a prefix to the new columns.
* **Concatenating DataFrames**: The original DataFrame is concatenated with the new dummy variables along the columns (axis=1).
* **Displaying the Result**: The updated DataFrame is printed, showing the original data along with the new binary columns for "gas" and "diesel".

Resulting DataFrame:

The resulting DataFrame will look like this:

car fuel\_type fuel\_diesel fuel\_gas

0 A gas 0 1

1 B diesel 1 0

2 C diesel 1 0

3 D gas 0 1

This shows how the categorical variable "fuel\_type" has been transformed into two binary features.

**Lesson Summary**

Congratulations! You have completed this lesson. At this point in the course, you know:

* Data formatting is critical for making data from various sources consistent and comparable.
* Master the techniques in Python to convert units of measurement, like transforming "city miles per gallon" to "city-liters per 100 kilometers" for ease of comparison and analysis.
* Acquire skills to identify and correct data types in Python, ensuring the data is accurately represented for subsequent statistical analyses.
* Data normalization helps make variables comparable and helps eliminate inherent biases in statistical models.
* You can apply Feature Scaling, Min-Max, and Z-Score to normalize data and apply each technique in Python using pandas’ methods.
* Binning is a method of data pre-processing to improve model accuracy and data visualization.
* Run binning techniques in Python using numpy's "linspace" and pandas' "cut" methods, particularly for numerical variables like "price."
* Utilize histograms to visualize the distribution of binned data and gain insights into feature distributions.
* Statistical models generally require numerical inputs, making it necessary to convert categorical variables like "fuel type" into numerical formats.
* You can implement the one-hot encoding technique in Python using pandas’ **get\_dummies** method to transform categorical variables into a format suitable for machine learning models.