**Data cleaning**

Motivation of data cleaning

1. To remove duplicates
2. To remove and fill the missing values
3. To engineer features with more standard or normalized forms
4. To tailor data into a type where specific model can process
5. Linear model: data under each feature need to have similar scales
6. Distance based: features with similar scales can be compared and measured under the same dimension
7. Tree based: it can handle unscaled continuous and categorical data.

Example data for this topic

A screenshot of a computer

Description automatically generated

* We convert the purchase date as daytime for later processing

**Duplicated Data**

* We can drop the duplicated data if the data is unique by unique identifier that can indicate this unique and same value appear elsewhere

A screenshot of a computer

Description automatically generated

* + By duplicated() method, we can find how many in the dataset values are duplicated or unique.
  + There is one duplicated observation and anything else is different from each other.
* We can visualize how they are the same to each other in both **entire observations**

A screenshot of a computer

Description automatically generated

* + .duplicated (keep = False) can enable to visualize the duplicated rows.
* We can drop the one of which rows and only keep the first row to remove duplicates.

A screenshot of a computer

Description automatically generated

* + .duplicated (keep = ‘first’) can remove all other duplicated rows and only keep the first row.
* **Duplicated data for subset of columns in observations**

A screenshot of a computer

Description automatically generated

* + Among each observation, there might be some duplicated columns subset while keeping other distinct columns.
  + Sometimes, those duplicated observations in particular columns are not necessary.
* Dropping duplicates 🡺 **data.drop\_duplicates( )**



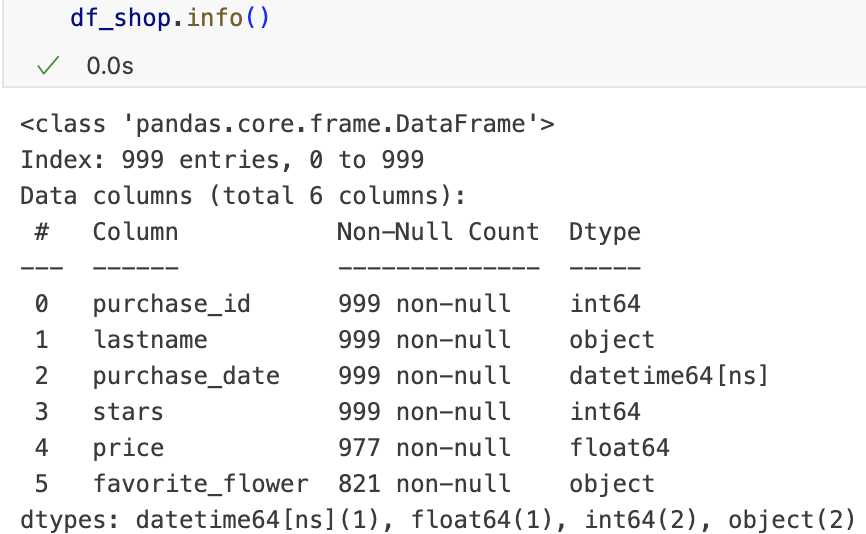
* + inplace = False will return a new dataframe without changing the original dataframe.
  + inplace = True will change the original dataframe.
  + keep = False will drop all duplicates.
  + keep = 'first' will keep the first occurrence of each duplicate.
  + keep = 'last' will keep the last occurrence of each duplicate.
  + subset = None will return and consider all columns in the dataset.

**Missing Data**

Several ways to deal with missing data

1. Drop rows
2. Impute data from the same column
3. Infer from other features
4. Fill with adjacent data

Use data.info to **check the missing value**



* Use data.isna ( ) or data.isnull ( ) will provide more detailed missing value information

A screenshot of a phone

Description automatically generated 🡺 .isna( ) = .isnull( )

* Note: data.isna() will only return an array of true or false of missing value in each observation 🡺 true = 1 and false = 0

A screenshot of a computer

Description automatically generated

* + To conceptualize the missing value of each columns, we can apply .sum( ) method to inform which observation contains the missing value and which is not.
    - Now, we know price and favorite flower columns contain the missing values with corresponding totals.
* To get to know the general structure of the missing value, we can use .sum( ).sum( ) to get the total # of the missing value in the dataset.

A close-up of a computer code

Description automatically generated

* Use negate logic to check rows with non-missing values

A screenshot of a computer code

Description automatically generated

* + .notna() is to check the non-missing value

**Drop rows for missing values 🡺 data.dropna()**

Original data info

A screenshot of a computer

Description automatically generated

1. If an observation contains at least 1 missing value (NaN), this row along with other info will be dropped as well.

A screenshot of a computer

Description automatically generated

* + It can ensure any rows with at least 1 missing value will be dropped from the dataset.

1. If we only want to drop missing value under a particular column and keep all other missing values in other column, we can specify the particular column by dropping the missing value

A screen shot of a computer

Description automatically generated

* + Any missing value under ‘price’ column is dropped.

1. We want to drop the row with all missing values in the data by ( how = ‘all’ )

A screen shot of a computer

Description automatically generated

* + The returned shape is the same as the original shape, meaning there is no row with all missing values.

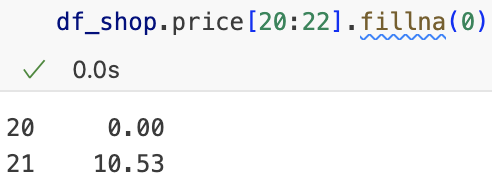
**Missing data manipulation**

Sometimes, we don’t want to lose the meaning of the missing values in the data by simply dropping them. Instead, we can grant them with the certain meaning.

1. Fill the missing value with constant (normally with 0 or 1)

* We can fill the price with 0 if it is missed 🡺 **data.fillna(0)**

A screenshot of a computer

Description automatically generated 🡺 

* Filling the missing value with constant with 0 will accidentally lower the overall column mean value, which may cause bad model interpretation.

A screen shot of a computer

Description automatically generated

1. Impute missing value with continuous data

***Data.fillna(Data.mean())***

* Fill the missing value with statistics of that column (mean, median, mode).

A screen shot of a computer code

Description automatically generated

* Filling missing value with mean will not change the overall feature characteristic too much based on the sample statistics.
  + Data.fillna(mean): it will fill the missing value with the mean of the column.
  + (Data.fillna(mean)).mean(): it will calculate the overall mean of that column after filling with the missing data with mean.

1. Impute missing value with categorical data.

***Data.fillna(Data.mode())***

* Fill with the mode to the categorical missing value.
* The missing data will be filled based on the most frequency category under that column feature.

A screenshot of a computer

Description automatically generated

* + ‘lilac’ is the data appears the most under the column.
* Count the number of the most frequent data

A screenshot of a computer

Description automatically generated

* Impute the missing value with the mode

A screenshot of a computer

Description automatically generated

* + The number of the ‘lilac’ is increased with imputation.

1. Simple Imputer

* mean**:** Replaces missing values with the mean of the column (*default for numerical data*).
* median**:** Replaces missing values with the median of the column (*useful for skewed data*).
* most\_frequent**:** Replaces missing values with the mode (*most frequent value*).
* constant**:** Replaces missing values with a constant value (requires fill\_value parameter).

A screenshot of a computer code

Description automatically generated

* + SimpleImputer (strategy = ‘mean’) indicates we will impute the missing value with mean of column ‘price’ and ‘stars’.
  + It will automatically refill the missing value after we fit the data to the column by .fit(data[[col1, col2]].
  + By .statistics\_, we can know what is the mean statistics of each column to be filled later.
* Fill the statistics to the missing value using .transform( )

A screenshot of a computer code

Description automatically generated

* + Missing values will be filled with continuous value
* For categorical value

A screenshot of a computer code

Description automatically generated

1. Infer method with linear regression model

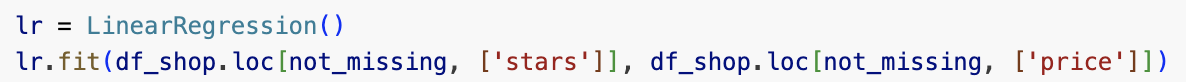
* Instead of imputing the missing data with statistics, we can use prediction from regression model.
* We want to fill the missing value to the price column based on the star column.
  + How to use **linear regression** to fill missing values in the ‘price’ column of a pandas DataFrame (df\_shop) based on the values in the ‘stars’ column

1. Identify the missing and non-missing value in the data.

A screenshot of a computer code

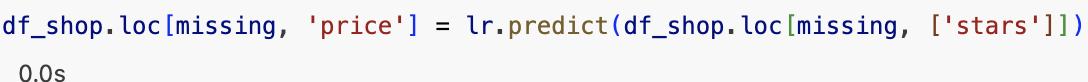
Description automatically generated

1. Initializing linear regression model



* + df\_shop.loc[not\_missing, ['stars']] **🡺 IV**
    - Extracts the stars column values for rows where the price column is **not missing**.
  + df\_shop.loc[not\_missing, ['price']] **🡺 DV**
    - Extracts the price column values for rows where price is **not missing**.

1. Predict missing value



* + df\_shop.loc[missing, ['stars']] **🡺 input value for prediction** 
    - Extracts the ‘stars’ column values for rows where the ‘price’ column is **missing**.
  + df\_shop.loc[missing, ‘price’] 🡺 output value for prediction
    - Based on the linear regression model between non-missing values between stars and price, now we train the model to predict those missing value on price based on its corresponding missing values on stars.
* It might cause **collinearity** since data will be correlated based on another features.

1. Use adjacent data to fill the missing data

* Typically, in time series problem, we can fill the missing value based on the surrounding data and decide to copy the data either on forward or backward filling.

A screenshot of a computer

Description automatically generated

* + In the data, we can determine which missing value will be filled based on direction.
    - Forward fill: propagate last valid observation forward to next valid

A screen shot of a computer code

Description automatically generated

* + - Backward Fill: use next valid observation to fill gap backwards

**Rescaling**

Rescaling feature is crucial for model training and distance comparison with same scale.

* Z-Score: standardization
* Min-Max rescaling

**Standardization**

* Use Z-Score to rescale the data to mean of 0 and standard deviation of 1.



* We want to rescale columns for ‘trip\_duration’ and ‘tip\_amount’

A screenshot of a computer

Description automatically generated

* Initiate standard scaler

A white background with blue text

Description automatically generated

* + We need to set and mean and SD for Z-score.
* Fit the standard scaler to the columns trip\_duration and trip\_amount.



* + We need to fit each data from 2 column to the standard scaler so that we can transform each data with the Z-score.
* Transform the data with Z-score for each column

A screenshot of a computer code

Description automatically generated

* Compare the statistics for original and Z-Scored data
  + For original statistics

A screen shot of a computer

Description automatically generated

* + For Z-Score statistics

A screenshot of a computer code

Description automatically generated

**Min-Max rescaling:** Scales features to a fixed range, usually [0, 1].

* Map minimum value to 0
* Map maximum value to 1

A black text on a white background

Description automatically generated

* The goal is to remove all negative values in the original dataset

1. Initialize the Min-Max rescale by fitting the data in the specified columns

A screenshot of a computer code

Description automatically generated

* + Set the range of rescaling between 0 and 1.
  + use .fit\_transform() to fit the columns and scale the columns with the function.

1. Compare the statistics between Z-score and Min-Max Scaler

A screenshot of a computer

Description automatically generated

**Skewness**

Data skewness will affect the data representation by having potential outliers.

* Use log transformation to reduce the skewness.
* If the data is heavy skewed, the model will be affected on training process.

1. The original distribution of total\_amount

A graph of a number of columns

Description automatically generated with medium confidence

1. Use square-root transformation of each value, leading data more normalized.

A graph of a graph

Description automatically generated

1. Use log transformation to the original data, delivering the normal distribution

A graph of a graph

Description automatically generated

**Outliers**

Similar to the missing data which doesn’t contribute too much meaning in the whole dataset.

1. Human data entry error
2. Instrument measurement errors
3. Data processing errors
4. Natural deviations

Outlier detection

* Outlier visualization
  + Boxplot and bar plot

A screenshot of a graph

Description automatically generated

Outlier reduction

* Drop outlier data
* Treat it as missing value
* Encode with dummy variable

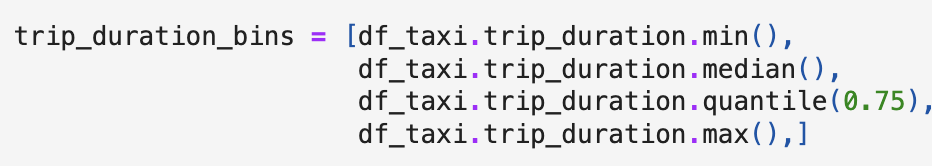
**Feature Engineering**

Feature engineering is a data manipulation strategy

**Binning 🡺 *pd.cut ()***

* Goal: continuous numerical variables are divided into discrete intervals, or “bins.”
  + This process transforms continuous data into categorical data
* Purpose:
  + **Handle Outliers**:
    - Binning can reduce the impact of extreme values by grouping data into broader categories.
  + **Capture Non-Linear Relationships**:
    - Binning can help detect and represent non-linear relationships between features and the target variable.
* Steps

1. Define Bin edges



* + Determine the interval based on the trip\_duration continuous variable.
    - df\_taxi.trip\_duration.min(): Smallest trip duration.
    - df\_taxi.trip\_duration.median(): Median trip duration (50th percentile).
    - df\_taxi.trip\_duration.quantile(0.75): 75th percentile of trip duration.
    - df\_taxi.trip\_duration.max(): Maximum trip duration.

1. Apply binnin with pd.cut ()

A computer code with black text

Description automatically generated

* + pd.cut ( ): Splits the trip\_duration column into bins defined by trip\_duration\_bins.
  + labels = ['short','medium','long']: Assigns category labels to each bin defined earlier in trip\_duration\_bins.
    - short: Smallest to median.
    - medium: Median to 75th percentile.
    - long: 75th percentile to max.
* note: any continuous values fall into any range of the label will be labelled as corresponding name.
  + right=True: Specifies that the bins are right-inclusive (e.g., [low, high]).
  + include\_lowest=True: Ensures the first bin includes the lowest value.

A screenshot of a table

Description automatically generated

**One-hot encoding 🡺 *pd.get\_dummies () or sklearn.preprocessing.OneHotEncoder***

* Goal: To convert categorical variables by representing each category as a separate binary feature.
* Create a numerical variable out of categorical variables when certain data is presented marked as ‘1’.
  + One column per category, ‘1’ in only one column per row.
* **How It Works**
  + For a categorical feature with N unique categories, one-hot encoding:
    - Creates N new binary columns (one for each category).
    - Assigns a value of 1 to the column corresponding to the category of the observation, and 0 to all others.
  + A screenshot of a black screen

    Description automatically generated
* **By pd.get\_dummies ()**
  + We can apply one-hot encoding after binning after we convert the categorical variables into binary variables

A screenshot of a computer

Description automatically generated

* + We can use prefix to tag the one-hot encoding columns.
* **By sklearn.preprocessing.OneHotEncoder**
  + In Sklearn, we do one-hot encoding after determining the binning

A screen shot of a computer code

Description automatically generated

* + We can fit the one-hot encoding with binning process previously determined and set them into categories for later categorization.

A screenshot of a computer program

Description automatically generated

* + - We fit and transform the data into one-hot encoding into the sparse dataset.
* **By sklearn.preprocessing.KBinsDiscretizer**
  + KBinsDiscretizer can perform the binning and one-hot encoding at the same time.

A screenshot of a computer code

Description automatically generated

* + - We can determine the # of the bins by n\_bins
    - We can determine which kind of encoding method by encode = ‘onehot’

**Converting ordinal value into the nominal 🡺 *data.map()***

* Map the categorical data with defined values that shows the ordinal manner

A screenshot of a computer

Description automatically generated

* + We map the M -> 1, L->2, XL-> 3 by data.map()
  + We define the ordinal mapping with dictionary.