Overview: Data Preparation

As stated previously in Week 2, there are two possible approaches we can use for Data Understanding:

Use guiding questions to understand the data with the input of domain experts or persons who have background knowledge of the data. We have learned this approach in Week 2.

Use software tool(s) to explore the characteristics of data. This approach also called data exploration uses software tool(s) to analyze the characteristics of the datasets, especially data quality and distribution of data.

In the Data preparation phase, data exploration is performed in many different phases in the data mining process, including Data Pre-processing or Data Preparation.

This week's topic will cover the Data Pre-processing phase and the second half of the Data Understanding phase through data exploration of a data mining process, as depicted in Figure 1.

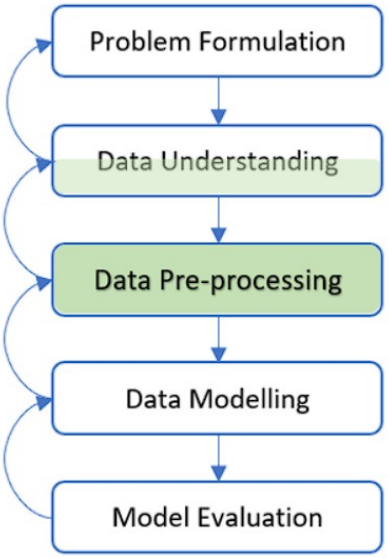


Figure 1. Data Understanding and Pre-processing

With preliminary examination, data exploration provides a high-level overview of each attribute in a dataset and the interrelationship with other attributes. Data

exploration answers questions like the normal values of an attribute, how much the data points differ from the normal value, or any outliers in the data set, for example. Data exploration can sometimes cover the entire data mining process. For example, scatterplots can determine clusters in low-dimensional data, which can further help develop preliminary classification models with simple visual rules.

Before applying data modeling algorithms or techniques in the Data Modeling phase of a data mining process, we need to preprocess data to prepare the data set to tackle any anomalies present in the data. Nevertheless, those anomalies need to be identified, which includes identifying outliers, missing values, and removal of duplicate or highly correlated attributes. For example, some data mining algorithms do not work well when input attributes as predictors are inter-correlated. Thus, it is vital to explore the data to identify them and remove them.

3.1 Data Preparation

Data preparation consists of two broad tasks:
i) data understanding to explore the available data using a software tool(s) and
ii) data pre-processing.

Data Understanding through Exploration

Data understanding through exploration includes:

- Using exploratory data analysis to familiarise with the data and discover initial insights.
- Evaluating the quality of the data.

Understanding the data characteristics and quality is essential before starting the data preparation task. Much of the raw data contained in databases is unprocessed and incomplete. For example, the datasets may contain:

- Attributes that are obsolete or redundant, or no longer relevant
- Missing values
- Outliers

- Misclassified data, which data values are inconsistent with policy or common sense.
- Data in a form not suitable for data mining models

More details on data understanding through exploration methods to understand their characteristics are described in Section 2.

Data Pre-processing

Data pre-processing includes:

- Preparing the initial raw data into the final dataset for all subsequent phases. This step is very labor-intensive.
- Select the attributes and records that are appropriate for the subsequent phases.
- If needed, perform treatment on specific attribute(s) due to quality issues such as missing values, outliers, and misclassification.
- Transforming certain attributes into different data types or values to suit the subsequent phases.
- Cleaning the raw data so that it is ready for the modeling algorithms.

Various methods are involved in the data preparation task to either treat data quality problems or prepare the data to suit the sequent phases.

Section 3.3 elaborates on the common data pre-processing methods for the respective purposes.

3.2 Data Understanding through Exploration

To understand the available datasets, there are two types of observation on data to explore:

1. Data Characteristics and
2. Data Quality.

We will learn the methods of both types in the following sections.

Data Characteristics

One of the purposes of performing data exploration is to understand the data characteristic. Therefore, it is essential that for each attribute in a dataset, we know the following:

- Dataset dimension: the number of instances (a.k.a. records or rows) and attributes (a.k.a. columns or features)
- Data domain: The data type of an attribute identified in the exploration tool and if the type and its values match the expectation based on the domain knowledge.
- Basic statistics of attributes: The statistics methods for understanding how the values in a dataset for an attribute are distributed across the value range they can take.

Data Quality

Another purpose of data exploration is to identify if the data have any data quality issues that could affect the models we build. The typical data quality issues include the following:

- Missing values: An instance (a.k.a, a record) containing an empty value for one or more attributes.
- Outlier: An instance of a dataset with an extremely low or high value for an attribute.
- Misclassification: An instance that has an inappropriate value for an attribute.

Some data quality issues arise due to invalid data and will be corrected as soon as we discover them. Others, however, occur due to perfectly valid data that may cause difficulty for some modeling techniques. We note these data quality issues during exploration for potential pre-processing when we reach the modeling phase.

Section 2.1 elaborates on data quality in more detail.

3.2.1 Understanding Data Quality

After exploring data to understand their characteristics and basic statistics, we further perform data exploration to identify if the data have any data quality issues that could affect the models we build.

Data quality is an essential factor in influencing the quality of the results from any analysis. In addition, data is often collected to answer specific questions or for various reasons. Therefore, data should be reliable and represent the defined subject.

The common data quality issues include:
1) Missing values,
2) Outliers,
3) Misclassification
4) Duplication.

Missing values

A missing value is an instance containing an empty value for one or more attributes.

There is always the chance that some data values have not been measured, are not answered, are unknown, or become lost during the completion of datasets. These values are called missing values.

Missing values are those that have not been measured, not answered, are unknown, or have become lost during the completion of datasets.

Missing data in the dataset for training a predictive model can reduce the model's performance or lead to a biased model because we have not correctly analyzed the behavior and relationship with other attributes. Consequently, data quality problems can lead to wrong predictions or classification.

Outliers

An outlier is a dataset sample with an extremely low or high value for an attribute. In other words, outliers are data with considerably different characteristics than most of the other samples in the dataset as diagrammatically elaborated in Figure 4.

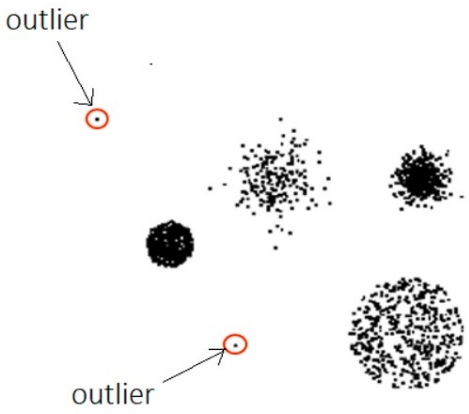


Figure 4. Outliers

There are many reasons for outliers. They can be categorized as follows:

- Measurement Error. This type of outlier existed when the measurement device used was faulty.
- Data entry Error. This type of outlier is caused by human errors such as mistakes made during data entry or collection.
- Experimental Error. This outlier is due to errors when planning or executing an experiment.
- Data Processing Error. When an error happens during data extraction or manipulation, it may cause a data processing error.
- Sampling Error. An outlier may happen when extracting or mixing data from the wrong or various sources.
- Intentional Outlier. This type of outlier is not due to error. It is a dummy outlier created to fit specific data mining solutions.
- Natural Outlier - Like Intentional outliers, natural outliers are not caused by an error. However, this type of outlier is not synthetic. Those that are not an outcome of a machine/human mistake or fabricated are called natural outliers.

Regardless of outliers, they can impact statistical analysis and machine learning by influencing the measures like mean and standard deviation. Consequently, outliers can bring bias to hypothesis tests and the performance of a predictive model.

Misclassification

Misclassification occurs when a particular attribute value(s) are assigned to a different level than the one they should be in. This misassignment can lead to incorrect associations between the given levels and the outcomes of the desired result.

To further understand misclassification data, we use the dataset view in Figure 5 as an example to identify various misclassification.

Level Name	Count
USA	1
France	1
US	156
Europe	46
Japan	51

Figure 5. Misclassification

Based on the dataset view, US ad USA is in the dataset, and both refer to the same meaning that represents the United States of America. Another misclassification data problem when we observe Europe exists in the dataset because it is a continent-level value. Still, the rest of the data values are at the country level.

Duplication

A dataset may include data that are duplicates or almost duplicates of one another. Duplication is a primary issue when merging data from heterogeneous sources. Examples of duplication can be the same data in multiple records.

3.3 Data Preparation Methods

Various methods are involved in the data preparation task to either treat data quality problems or prepare the data to suit the subsequent data mining phases.

Methods to treat Missing Values

1. Deletion
    a) If an attribute contains a lot of missing values, consider removing the attribute
    b) If only a few examples contain missing values, consider removing those cases/rows
2. Imputation
    a) We can introduce a new category in a categorical attribute with missing values, for example, "unknown" or "U" to denote the intended meaning. Besides, we can also use the Mode or Median to replace the missing values.
    b) We can consider using Mean for a numerical attribute to replace the missing values.
3. Prediction model
    a) It is a sophisticated method for handling missing data. Here, we create a predictive model to estimate values that substitute the missing values.
4. Do nothing
    a) It may be possible to leave the missing values null if the null represents a specific meaning. For example, the payment transaction date of a newly registered customer may be null.

Methods to handle Outliers

1. Deleting rows of observation
    a) We delete outlier values due to a data entry error, data processing error, or outlier observations that are very small in numbers. We can also use trimming at both ends to remove outliers.
2. Transforming the data values
    a) Using a Natural log or other mathematical formulas such as Square Root, Range, or Z transformation on a value reduces the variation caused by extreme values of outliers.
    b) Binning or grouping is also a form of variable transformation.
3. Do nothing

a) It may be possible to leave the outliers as they are, especially when the data mining problem is aimed at identifying abnormal events that are actually outliers.

Methods to treat Misclassification

The common method to correct misclassified data is to update the data according to the correct level.

Methods to treat Duplication

Performing deletion is usually the common method to eliminate duplication(s) that contain the same data in multiple records.

Methods for Variable Creation

It is a process of generating a new attribute based on an existing attribute(s). For example, date (dd-mm-yy) is an input variable in a data set. We can generate new variables like day, month, year, week, and weekday that may better correlate with the target variable. Another example is generating a new attribute for Age deriving from a birthdate. This step helps underscore the hidden relationship in an attribute. There are two types of variable creation, i) Derived attributes and ii) Dummy attributes.

1. Creating derived attributes
    a) It refers to creating new attributes from existing attribute(s) values using a set of functions.
    b) We can also use transformation methods such as taking a log of attribute values, binning attributes, and other transformation methods.
2. Creating dummy attributes
    a) The most common application of dummy attributes is to convert a categorical variable into numerical variables, such as 0 and 1.
    b) Dummy attributes are also called Indicator Variables.

Selecting Relevant Attributes

Datasets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant. Although it may be possible for a domain expert to select some relevant attributes, it can be difficult and time-consuming. Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, creating a bias for employed data modeling algorithms and likely slowing down the mining process.

Removing irrelevant attributes is the common method for selecting relevant attributes.

3.4 Case Study: Customer Churn

We return to the customer churn case study datasets to practically apply the methods for data exploration and pre-processing.

We will perform data understanding to ensure the data types are correct according to the domain knowledge. It is critical to ensure the right data types because they may present different meanings. Before exploring the basic statistics of the datasets, data preparation is necessary to fix the wrongly defined data types so that basic statistics can be derived correctly.

The following sections will apply data preparation methods to our customer churn data set.

3.4.1 Discussion Forum Activity: Understanding Data Characteristics

For this course, we will learn how to explore data characteristics using Python software as the tool running on the Ed platform. We will use the customer churn datasets to exercise the exploration methods practically. There are three initial raw datasets as described in the metadata presented in Week 2:

Customer: stored in customer.xlsx file under the /data/ folder.

Churn: stored in the churn.xlsx file under the /data/ folder.

Payment transaction: stored in the ptransaction.xlsx under the /data/ folder.

We have learned the following characteristics in the previous section, and now we will use Python programming codes to implement the data exploration method. In the Python programming environment, we must first access the datasets before exploring the data. The following codes give examples of accessing the datasets we want to explore, via DataFrame using the pandas library to contain the datasets.

import pandas as pd

```
df1 = pd.read_excel('data/customer.xlsx')
df2 = pd.read_excel('data/churn.xlsx')
df3 = pd.read_excel('data/ptransaction.xlsx')
```

Copy and paste the above codes to the `main.py` file and run the program by entering the `python main.py` command on the Terminal interface.

This course focuses on the concepts of the data mining process and uses a case study to apply and implement concepts using the Python software tool. Explanation of Python programming concept is beyond the scope of this course. However, students should have the knowledge and skills in Python programming as the pre-requisite of this course specifically for the concept implementation.

Dataset Dimension:

To understand a dataset dimension, we explore the data to observe the number of instances (a.k.a. records or rows) and attributes (a.k.a. columns or features). Add the following Python codes after the existing lines (or you can always start the new lines after a few empty lines to the main.py file.

```
# to display dataset dimension in row and column
print("Customer dataset: ", df1.shape)
print("Churn dataset: ",df2.shape)
print("Payment transaction dataset: ",df3.shape)
```

Run the `main.py` program. Observe the dimension of the datasets.

The shape function displays the number of rows (a.k.a records or samples) and columns (a.k.a. data fields or attributes) of the respective datasets.

After running the program, we should observe, for example, that the Customer dataset contains 1000 records and six attributes and the Churn dataset has only one attribute with 503 records.

Data Domain:

To get the information of an attribute's data type and check if the type and its values match the expectation based on the domain knowledge. We can use the info() function as follows:

```
# to display dataset column names, #non-null, data type
print(df1.info())
print(df2.info())
print(df3.info())
```

Copy and paste the above codes to observe the exploration results.

To understand the results, take the example of the Customer dataset, it should be observed that it contains 8 attributes (i.e., customerId, Firstname, Gender, PostalCode, HashCode, Birthdate). All the attributes do not contain any NULLvalues (i.e. indicated with 1000 non-null) except the Birthdate and Gender attributes have a record of having a NULL value, which, conversely, 999 records have a birth date value. We do not know why there is a NULL value in both attributes may be due to an error or no available information at the time of data entry. To deal with this NULL value, we will study the possible methods in the next section of Data Pre-processing Methods.

The info() function also represents the data type information, for example, the Customer dataset has the data types of attributes as follows:

```
...[results above are ignore here]...
dtypes: datetime64[ns](1), int64(2), object(3)
...[results below are ignore here]…
```

The result above shows that there is one attribute (Birthdate) having DateTime type, two (PostalCode and CustomerId) integers (i.e. numbers), and three (Firstname, Gender, and HashCode) objects. If we refer to the metadata of the datasets described in the Week 2 topic for data understanding based on domain knowledge, we discover that there are unmatched data types. The following table shows the matched and unmatched data types according to domain knowledge:

Dataset: **Customer**

Attribute Name	Data Type (Domain knowledge)	Data Type (data exploration)	Data Understanding Finding
CUSTOMERID	Numbers	Numbers	Matched data type
FIRSTNAME	Characters	Object	Unmatched data type
POSTALCODE	Characters	Numbers	Unmatched data type
HASHCODE	Characters	Object	Unmatched data type
BIRTHDATE	Date	Date	Matched data type
GENDER	Characters	Object	Unmatched data type

Figure 2. Exploring and Verifying Data Types based on Domain Knowledge

Observe the Churn and Payment transaction datasets' attributes and their data types. Refer to the metadata presented in Week 2, and use the example codes given earlier to identify the unmatched data types as presented in Figure 2.

Time: 30 minutes

Purpose: This activity aims to use the example codes based on metadata presented in Week 2 to identify the unmatched data types as presented in Figure 2.

Task: Refer to the metadata presented in Week 2, and use the example codes given earlier to identify the unmatched data types in Figure 2. Share your responses on the Discussion Board.

Feedback: Feedback to your posts in the Discussion Board will be provided as a reply to your comments.

3.4.2 Data Preparation for Integrating Sources

This section teaches how to integrate multiple data sources to obtain more dimensions for analysis and modeling. To practically learn how to implement the methods, we return to the mini-case study of customer churn. We have observed three available datasets. The following summarizes the datasets, their source file

name, and the unique key (i.e., the attribute that its value can uniquely identify a row of a dataset)

Customer (customer.xlsx): the unique key is the CustomerId attribute.

Churn (churn.xlsx): the unique key is the CustomerId attribute.

Payment transaction (ptransaction.xlsx): the unique key is the CustomerID attribute.

Based on the unique key of each dataset of Customer, Churn, and Payment transaction, we can join them together using the CustomerId attribute.

Identifying and Removing Duplication

Before joining the datasets, it is essential to check if duplications exist to avoid redundancy. The following example of Python codes can be used to find duplications with the duplicated() function.

```
# import library and access data sources
import pandas as pd
import math
from datetime import datetime, timedelta

df1 = pd.read_excel ('data/customer.xlsx')
df2 = pd.read_excel ('data/churn.xlsx')
df3 = pd.read_excel ('data/ptransaction.xlsx')

# identify if duplication exists
print(df1[df1.duplicated()])
print(df2[df2.duplicated()])
print(df3[df3.duplicated()])
```

If there are duplications, then we can use the following example codes to remove the duplications using the drop_duplicates() function to find duplicates and the len() to count the number of duplicates that exist. If none, then proceed to the data types exploration.

```
# if duplication exists, remove and keep only one
if len(df1[df1.duplicated()]) > 0:
```

```
    df1 = df1.drop_duplicates(keep = 'first')
if len(df2[df2.duplicated()]) > 0:
    df2 = df2.drop_duplicates(keep = 'first')
if len(df3[df3.duplicated()]) > 0:
    df3 = df3.drop_duplicates(keep = 'first')
```

Identifying and Converting Unmatched Data Types

As we have observed the data type of each attribute in the Customer dataset in the previous section 4.1, four attributes' data types need to be corrected. The Firstname, HashCode, PostalCode and Gender should be set as string type. The following Python commands serve the conversion for correction.

```
# convert data type
df1['Firstname'] = df1['Firstname'].astype('string')
df1['PostalCode'] = df1['PostalCode'].astype('string')
df1['HashCode'] = df1['HashCode'].astype('string')
df1['Gender'] = df1['Gender'].astype('string')
```

Creating a New Attribute to Indicate the Target 'Churn'

The Churn dataset contains all the CustomerId of customers that have already churned but do not have an attribute to indicate the classification target, i.e. Churn or not Churn. For this case study, we use the value of 1 to indicate churn, else 0.

To have an indicator of customer churn, we create a new attribute called Churn (i.e. Dummy attribute) as the indicator with a value equal to 'yes' denotes a churned customer. Next, declare the data type of the new attribute to the character type using the astype('string') function.

```
# create a dummy variable, naming it Churn attribute to indicate
churn with a value 'yes'
df2['Churn'] = 'yes'
df2['Churn'] = df2['Churn'].astype('string')
```

Creating a New Derived Attribute for Age

In the Customer dataset, we observe the Birthdate attribute. However, the attribute carries a hidden meaning of age. Therefore, creating and deriving a new attribute Age, to represent age better reflects the meaning. Because the case study data was collected until the end of February 2022, we calculate a customer's age based on this timeline. The following codes show the formula of age calculation according to the timeline.

```
# create Age Attribute deriving from Birthdate.
df1['Age'] = (datetime(2022, 3, 1) - df1['Birthdate']) //
timedelta(days=365.2425)
```

Integrating Data Sources

We can use the Python merge() function to integrate all data sources. We perform a left join to merge datasets for the implementation by ensuring all rows from the Customer dataset are captured. We want to perform churn classification modeling based on customers' profiles.

```
#returns all rows from the Customer, even if there are no matches in
Churn dataset
df_merge = df1.merge(df2, on='CustomerId', how='left')

#returns all rows from the Customer, even if there are no matches in
Transaction
df_merge = df_merge.merge(df3, on='CustomerId', how='left')
```

Further Pre-processing to ensure Data Quality

To further prepare the merged dataset, there several pre-processing tasks are necessary. Observe the missing values of each attribute after the merge using the info() function.

```
# observe the dataset attributes and data types
print(df_merge.info())
```

Based on the info() result, several attributes have missing values. Next, we examine how these missing values can be treated according to the meaning of each attribute:

When we created a dummy variable, i.e., the Churn attribute in the Churn dataset, we only captured those customers who have already churned. During merging between Customer and Churn datasets, only customers who have churned contain a value of 1. Those not churned have a null value. We will replace the null or missing value with 0, indicating the non-churn status.

The payment transaction dataset does not contain all customers' payment details, i.e., some customers do not have payment transactions. Also, we observe that one Customer dataset row does not contain gender information. Consequently, null or missing values will exist in the merged dataset. To tackle this problem, we can either delete those rows having missing values or replace the null data with a value. Based on the problem understanding, the classification requires understanding customers' demographics and payment transactions' impact on churn. Thus, we will delete the rows that have missing transaction data.

Identifying and Treating Missing Values

Besides the info() function, we can use another option to explore the existence of the missing values in a specific attribute using the isna() function. The example codes are as follows.

```
for i, col in enumerate(df_merge.columns):
    print(col, df_merge[col].isna().sum())
```

Comparing the info() and isna() functions, info() gives the number of not null values and information on the data type of each attribute in a dataset, whereas isna() simply shows the number of missing values that exist in each attribute.

We perform the following tasks based on the rationale for pre-processing using the fillna() function to identify missing values/null and fill in a value into a specific attribute's data.

We use the apply(lambda x : math.floor(x) function to set the data type of the attributes to an integer, and use a 'no' value for the Churn attribute to replace the missing values for those customers who have not churned.

```
# to treat null or missing values
df_merge['Churn'] = df_merge['Churn'].fillna('no')
df_merge['Cash'] = df_merge['Cash'].fillna(0)
df_merge['Cash'] = df_merge['Cash'].apply(lambda x : math.floor(x))
df_merge['CreditCard'] = df_merge['CreditCard'].fillna(0)
df_merge['CreditCard'] = df_merge['CreditCard'].apply(lambda x :
math.floor(x))
df_merge['Cheque'] = df_merge['Cheque'].fillna(0)
df_merge['Cheque'] = df_merge['Cheque'].apply(lambda x :
math.floor(x))
```

We have now treated missing values with a value of 0 to reflect their meaning in the respective attributes. For the rest of the attributes containing missing values, such as Gender, LastTrxDate, and SinceLastTrx (derived from the LastTrxDate), we delete the rows because replacing them with any value in the data is meaningless.

We use the dropna() function to drop all the rows with at least one missing in any attribute.

```
# to delete the remaining rows with missing values in any attribute
df_merge.dropna(inplace=True)
```

Performing Data Transformation using Replacement

In a real-world scenario, a postal code represents a specific area. For example, we may use the PostalCode to identify a customer's living area. However, directly taking the original postal code with many levels (i.e., different values) in the Customer dataset may cause the classification model's ability to generalize the future data for prediction. This problem is called overfitting.

We will look at the overfitting problem in mode detail in Week 6.

To avoid overfitting problems due to too many levels in the PostalCode attribute, we use the first character of the postal code to analyze a broader area instead of discarding the information totally. To realize this deployment, we replace the original postal code with the first character of the value.

To select the first character of the attribute, we use the following Python code to implement the selection and set it as a string type.

```
# to select the first character of the PostalCode
df_merge['PostalCode'] =
df_merge['PostalCode'].str[0].astype('string')
```

Selecting Relevant Attributes

The previous steps have integrated all data sources and prepared the attributes to reflect the correct meaning and data types. However, some irrelevant such as HashCode, represent a meaningless computer-generated value. Moreover, a duplicated attribute such as Birthdate that the Age has better reflects the meaning. Besides, the LastTrxDate values have been derived and reflected in the SinceLastTrx attribute. Thus, we will exclude these three attributes and select the rest of the other attributes.

We use the following codes to implement the selection.

```
# to select useful attributes for further analysis and modeling
df_merge = df_merge[['CustomerId','Gender', 'Age','PostalCode',
'MinTrxValue', 'MaxTrxValue', 'TotalTrxValue', 'Cash', 'CreditCard',
'Cheque', 'SinceLastTrx', 'Churn']]
```

Saving the Consolidated Data

We have now completed the data preparation tasks for the case study to bring the consolidated data for further analysis. Observe the final dataset dimension, attributes, and data types to confirm their structure for further analysis.

```
# observe the consolidated data dimension and data types
print(df_merge.shape)
print(df_merge.info())
```

Finally, we save the data into a new and final dataset file using the following example Python code, naming it CustomerChurn.csv in a CVS format.

```
# save the data into a cvs file
df_merge.to_csv('CustomerChurn.csv')
```

Observe the file directory, the CustomerChurn.csv should be available for download and view.

3.4.3 Understanding Data Statistics for Pre-processing

The statistics methods for understanding how the values in a dataset for an attribute distribute across the value range they can take are skewness, mean, min, max, and mode.

Exploring Numerical Attributes' Data

To explore attributes with numerical data types, for example, using the describe() function to observe the basic statistics of all numerical attributes.

We can also use statistical functions such as the skew() function to observe a specific attribute's skewness. Returning to the merged dataset, CustomerChurn, we can use the following codes for exploring data statistics:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
```

```
df = pd.read_csv('data/CustomerChurn.csv')

# show basic statistics for numerical columns (count, mean, std, min,
max, 25%, 50%, 75%)
print(df.describe())

# show skewness and kurtusis of a particular columns
print('-----------Skewness--------------')
print(df.skew())
```

Generally, the skewness for a normal distribution is zero, and any symmetric data should have a skewness near zero. However, based on the skew() function results, we observe that the MinTrxValue, MaxTrxValue, TotalTrxValue, CreditCard, and Cheque attributes have a skewed distribution with skewness values not near zero when rounded to the numbers.

There are many more statistical functions available in the Python programming environment. In addition, students can try out other functions as additional learning activities.

As we understand from the metadata described in Week 2, the Cash, CreditCard, and Cheque attributes' data value of 1 indicate if a customer used to pay by cash, credit card, or cheque, or 0. Therefore, since they only contain two values, we ignore the skewness findings but further explore to confirm if their values should fall within 0 or 1.

In the next Section 4.4, we will use the above skewness findings to explore further the data of the the MinTrxValue, MaxTrxValue, TotalTrxValue, CreditCard, and Cheque attributes to detect if outliers exist.

Exploring Categorical and Binary Attributes' Data

The Python function value_counts() shows the counts of each value of an attribute. We can also use the value_count() to identify if specific attribute data fall within the expected values or range.

To explore attributes with character data types (i.e., categorical), for example, using the following Python functions to observe the categorical attributes Gender, PostalCode, and Churn, we can use the following codes:

```
# show the values exist in the attribute and their counts
print(df['Gender'].value_counts())
print(df['Churn'].value_counts())
print(df['PostalCode'].value_counts())

# further exploration to check values
print(df['Cash'].value_counts())
print(df['Cheque'].value_counts())
print(df['CreditCard'].value_counts())
```

After running the above codes, we can observe that the Gender attribute contains four values, male (496 rows), female (400 rows), männlich (51 rows), and weiblich (49 rows). As we learned that the original datasets were collected from Germany, the German terms männlich and weiblich refer to males and females, respectively. Since they have the same meaning, we should fix this misclassification problem.

The Churn, PostalCode, Cheque, and CreditCard attributes contain the data falling within the expected values. We expect the PortalCode attribute data values to fall within '1' to '9', the Churn attribute to have 'yes' or 'no', and both CreditCard and Cheque to have 0 or 1. Therefore, no further pre-processing is necessary for them.

Treating Misclassification

To tackle the misclassification problem that exists in the Gender attribute, we will replace the value of männlich value to male, and weiblich to female using the following Python codes:

```
# to locate männlich in Gender attribute and replace it with male
df.loc[df['Gender'] == 'männlich', 'Gender'] = 'male'
```

```
# to locate weiblich in Gender attribute and replace it with female
df.loc[df['Gender'] == 'weiblich', 'Gender'] = 'female'
To confirm the misclassification has been fixed, we observe the
Gender attribute data again:

# to confirm the Gender attribute values are correct
print(df['Gender'].value_counts())
```

Saving the Data after Pre-processing

After treating the misclassification problem in the Gender attribute, we save the data and name it ChurnProcessed.csv.

```
# save the data into a CSV file
df.to_csv('ChurnProcessed.csv')
```

3.4.4 Pre-processing for Treating Outliers

Outliers are the extreme values within the dataset when exploring data. In other words, the outlier data points vary significantly, either much larger or smaller than the expected values.

For the following data exploration to detect outliers, we will use the dataset that contains pre-processed data after the tasks performed in previous sections, stored in the CSV file ChurnProcessed.csv.

```
import pandas as pd
df = pd.read_csv('data/ChurnProcessed.csv')
```

In Section 4.3, we obtained the following findings regarding the customer churn processed dataset:

- the MaxTrxValue, MinTrxValue, and TotalTrxValue attributes have a skewed distribution that requires treatment. Further exploration is needed to detect if outliers exist.

- the Age and SinceLastTrx attributes have a normal distribution. Further exploration is needed to detect if outliers exist.
- the rest of the attributes have data that falls within the values or range. No further data exploration is required.

Detecting Outliers

There are several methods we can use to identify outliers that exist in a numerical attribute. The selection of methods depends on the distribution of the data. We are going to learn the common methods for finding outliers and practically observing the data and outliers using Python codes.

1. For Normal Distributions

In the case of normal distribution, the data points which fall below or above the derived value of the following formula are outliers:

```
mean−3* (standard deviation)
```

where the mean and the standard deviation s the average value of a particular attribute.

The Python codes below find the mean and standard deviation for deriving the acceptable values range. The statements with the # prefix comment on the respective lines of codes. We apply this method to the Age and SinceLastTrx attributes since they both have a normal distribution.

```
#get acceptable range h - l
# find the highest value allowed h
h1 = df['Age'].mean() + 3*df['Age'].std()
h2 = df['SinceLastTrx'].mean() + 3*df['SinceLastTrx'].std()

# find the lowest value allowed l
w1 = df['Age'].mean() - 3*df['Age'].std()
w2 = df['SinceLastTrx'].mean() - 3*df['SinceLastTrx'].std()


# get outliers based on the range identified between h and l
```

```
df1_outliers = df[(df['Age'] > h1) | (df['Age'] < w1)]
df2_outliers = df[(df['SinceLastTrx'] > h2) | (df['SinceLastTrx'] <
w2)]
# find the number of outliers
print('No. of outliers (mean-std) in Age = ', df1_outliers.shape[0])
print('No. of outliers (mean-std) in SinceLastTrx = ',
df2_outliers.shape[0])
```

2. For Skewed Distributions

In the case of skewed distributions, we use Inter-Quartile Range (IQR) proximity rule, in which data points that fall below the Q1 – 1.5 IQR or above Q3 + 1.5 IQR are outliers.

Q1 and Q3 are the 25th and 75th percentile of the explored dataset, respectively, and IQR denotes the inter-quartile range of Q3 – Q1.

We apply this method to the the MaxTrxValue, MinTrxValue, and TotalTrxValue attributes since they have a skewed distribution.

```
# find the upper/lower limits for each attribute
pct25_ttl = df['TotalTrxValue'].quantile(0.25)
pct75_ttl = df['TotalTrxValue'].quantile(0.75)
pct25_max = df['MaxTrxValue'].quantile(0.25)
pct75_max = df['MaxTrxValue'].quantile(0.75)
pct25_min = df['MinTrxValue'].quantile(0.25)
pct75_min = df['MinTrxValue'].quantile(0.75)

# calculate the iqr and range for each attribute
iqr_ttl = pct75_ttl - pct25_ttl
up_ttl = pct75_ttl + 1.5 * iqr_ttl
low_ttl = pct25_ttl - 1.5 * iqr_ttl
iqr_max = pct75_max - pct25_max
up_max = pct75_max + 1.5 * iqr_max
low_max = pct25_max - 1.5 * iqr_max
iqr_min = pct75_min - pct25_min
up_min = pct75_min + 1.5 * iqr_min
low_min = pct25_min - 1.5 * iqr_min
```

```
# detect and print number of outliers in each attribute
df3_outliers = df[(df['TotalTrxValue'] > up_ttl) |
(df['TotalTrxValue'] < low_ttl)]
print('No. of outliers (IQR) in TotalTrxValue = ',
df3_outliers.shape[0])
df4_outliers = df[(df['MaxTrxValue'] > up_max) | (df['MaxTrxValue'] <
low_max)]
print('No. of outliers (IQR) in MaxTrxValue = ',
df4_outliers.shape[0])
df5_outliers = df[(df['MinTrxValue'] > up_min) | (df['MinTrxValue'] <
low_min)]
print('No. of outliers (IQR) in MinTrxValue = ',
df5_outliers.shape[0])
```

After running the above Python codes, we found that the TotalTrxValue, MaxTrxValue, and MinTrxValue attributes contain outliers.

To further explore these three attributes to gain a clearer picture of their outliers, we can plot a histogram for each of these attributes to observe the data. We use the following example Python codes to plot the charts:

```
import matplotlib.pyplot as plt
import seaborn as sb

plt.figure(figsize=(10,15)) # define the plot width and height

# subplot(x, y, z) where x=rownum, y=colnum, z=plotnum, x(y)must
be >=z
plt.subplot(3,1,1)
sb.distplot(df['MinTrxValue'])
plt.subplot(3,1,2)
sb.distplot(df['MaxTrxValue'])
plt.subplot(3,1,3)
sb.distplot(df['TotalTrxValue'])

plt.savefig('plot_dist.png')
```

The subplot() function defines the dimension of the plot and the distplot() function presents a specific attribute data distribution.

After running the above codes, observe the content in the plot_dist.png file saved in the directory.

Treating Outliers

Based on the plots in plot_dist.png, we observe that the TotalTrxValue, MaxTrxValue, and MinTrxValue attributes have a wide range of values (i.e., negative to 1000 scaling points). There are several methods to treat outliers. We use the square root transformation for this case study by applying the numpy.sqrt() function to the data to reduce the scale while treating the outliers.

Students can try to explore other different methods such as the log transformation to treat outliers.

The computed values are stored in new attributes with the prefix “Sqrt” to derive the square root values of the TotalTrxValue, MaxTrxValue, and MinTrxValue attributes.

```
import numpy
# apply sqrt on attributes that have outliers
df['SqrtTotal'] = np.sqrt(df['TotalTrxValue'])
df['SqrtMax'] = np.sqrt(df['MaxTrxValue'])
df['SqrtMin'] = np.sqrt(df['MinTrxValue'])
```

After applying the data transformation method, we can observe that the values range has been reduced using the following example codes.

```
print ('SqrtTotal min/max: ', df['SqrtTotal'].min(),
df['SqrtTotal'].max())
print ('SqrtMax min/max: ', df['SqrtMax'].min(), df['SqrtMax'].max())
print ('SqrtMin min/max: ', df['SqrtMin'].min(), df['SqrtMin'].max())
```

Saving the Data after Pre-processing

After treating the outliers problem, we save the data and name it ChurnFinal.csv.

This newly saved dataset contains additional attributes of SqrtTotal, SqrtMax, and SqrtMin.

```
# save the data into a CSV file
df.to_csv('ChurnFinal.csv')
```

Activities

Use the dataset(s) that the student has chosen in Week 2 Activity 1, and perform the following tasks:

1. Exploring data to understand their characteristics;
2. Exploring data to detect if there is a data quality problem(s);
3. Preprocessing data to integrate data sources (if applicable);
4. Preprocessing data to treat the quality problem(s);
5. Save the data to a final dataset.

Practice Questions

Question 1:

X is an attribute that should contain either 'female' or 'male' or 'unknown' value.

Observe the following partial data view of X:

What kind of data quality problem is X having?

X
female
male
unknown
m
female

ANS: Misclassification

Question 2:

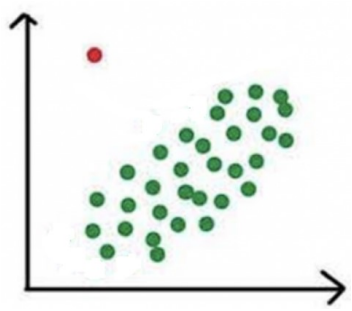
Data exploration using software tool(s) helps to understand and detect:

ANS:

- Data Quality
- Data dimension
- Data distribution

Question 3:

Observe the following data plot:



What are the possible methods to treat the red-color data?

ANS:

- Do nothing
- Deletion
- Log transformation
- Grouping