# CHAPTER 11: Data Preparation

## Introduction

In the previous chapter, you saw how critical it was to get a very good understanding of your data and learned about different techniques and tools to achieve this goal. While performing Exploratory Data Analysis (EDA) on a given dataset, you may find some potential issues that need to be addressed before the modeling stage. This is exactly the topic that will be covered in this chapter. You will learn how you can handle some of the most frequent data quality issues and prepare the dataset properly.

This chapter will introduce you to the issues that you will encounter frequently during your data scientist career (such as duplicated rows, incorrect data types, incorrect values, and missing values) and you will learn about the techniques you can use to easily fix them. But be careful – some issues that you come across don't necessarily need to be fixed. Some of the suspicious or unexpected values you find may be genuine from a business point of view. This includes values that crop up very rarely but are totally genuine. Therefore, it is extremely important to get confirmation either from your stakeholder or the data engineering team before you alter the dataset. It is your responsibility to make sure you are making the right decisions for the business while preparing the dataset.

## Handling Row Duplication

Most of the time, the datasets you will receive or have access to will not have been 100% cleaned. They usually have some issues that need to be fixed. One of these issues could be duplicated rows. Row duplication means that several observations contain the exact same information in the dataset. With the pandas package, it is extremely easy to find these cases.

In this session, we will be working with cancer data set.

Importing dataset into a DataFrame.

**Graphical user interface, text, application

Description automatically generated**

****

The duplicated() method from pandas checks whether any of the rows are duplicates and returns a Boolean value for each row, True if the row is a duplicate and False if not:



You should get the following output:

Table

Description automatically generated

In Python, the True and False binary values correspond to the numerical values 1 and 0, respectively. To find out how many rows have been identified as duplicates, you can use the sum() method on the output of duplicated(). This will add all the 1s (that is, True values) and gives us the count of duplicates:

****

You should get the following output:

****

Since the output of the duplicated() method is a pandas series of binary values for each row, you can also use it to subset the rows of a DataFrame. The pandas package provides different APIs for subsetting a DataFrame, as follows:

**Text

Description automatically generated**

The first API subsets the DataFrame by row or column. To filter specific columns, you can provide a list that contains their names. For instance, if you want to keep only the variables, that is, 'avgAnnCount', 'avgDeathsPerYear', and 'TARGET\_deathRate', you need to use the following code:

A picture containing diagram

Description automatically generated

You should get the following output:

Table

Description automatically generated

If you only want to filter the rows that are considered duplicates, you can use the same API call with the output of the duplicated() method. It will only keep the rows with True as a value:



You should get the following output:

Table

Description automatically generated

If you want to subset the rows and columns at the same time, you must use one of the other two available APIs: .loc or .iloc. These APIs do the exact same thing but .loc uses labels or names while .iloc only takes indices as input. You will use the .loc API to subset the duplicated rows and keep only the selected four columns, as shown in the previous example:

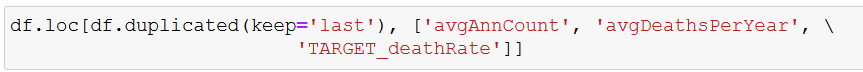


You should get the following output:

Table

Description automatically generated

This preceding output shows that the first few duplicates are row numbers 43, 44, 45, and so on. By default, pandas doesn't mark the first occurrence of duplicates as duplicates: all the same, duplicates will have a value of True except for the first occurrence. You can change this behavior by specifying the keep parameter. If you want to keep the last duplicate, you need to specify keep='last':

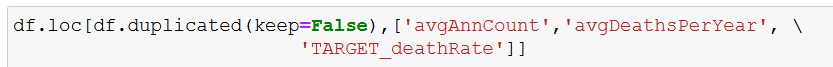


You should get the following output:

Table

Description automatically generated

As you can see from the previous outputs, row 35 has the same value as row 43. As expected, row 43 is not marked as a duplicate anymore. If you want to mark all the duplicate records as duplicates, you will have to use keep=False:



You should get the following output:

Table

Description automatically generated

This time, rows 35 and 43 have been listed as duplicates. Now that you know how to identify duplicate observations, you can decide whether you wish to remove them from the dataset. As we mentioned previously, you must be careful when changing the data. You may want to confirm with the business that they are comfortable with you doing so. You will have to explain the reason why you want to remove these rows. In the Cancer dataset, if you take rows 35 and 43 as an example, these two observations are identical.

In this case, you know that you shouldn't remove these rows. When you’re talking with the business, they may tell you that duplication shouldn't happen and that it may be due to human error as the data was entered or during the data extraction step. Let's assume this is the case; now, it is safe for you to remove these rows.

To do so, you can use the drop\_duplicates() method from pandas. It has the same keep parameter as duplicated(), which specifies which duplicated record you want to keep or if you want to remove all of them. In this case, we want to keep at least one duplicate row. Here, we want to keep the first occurrence:



You should get the following output:

Table

Description automatically generated

The output of this method is a new DataFrame that contains unique records where only the first occurrence of duplicates has been kept. If you want to replace the existing DataFrame rather than getting a new DataFrame, you need to use the inplace=True parameter.

The drop\_duplicates() and duplicated() methods also have another very useful parameter: subset. This parameter allows you to specify the list of columns to consider while looking for duplicates. By default, all the columns of a DataFrame are used to find duplicate rows. Let's see how many duplicate rows there are while only looking at the 'avgAnnCount', 'avgDeathsPerYear', 'TARGET\_deathRate' columns:

You should get the following output:



## Exercise 11.01: Handling Duplicates in a Life Expectancy Data

In this exercise, you will learn how to identify duplicate records and how to handle such issues so that the dataset only contains unique records. Let's get started:

Note

The dataset that we're using in this exercise is the Breast Cancer Detection dataset,

This dataset can be found in this book's GitHub repository:

1. Importing the pandas package:



Importing the Dataset Using the read\_csv() method from the pandas package, load the dataset into a new variable called df.

Chart

Description automatically generated

1. Display the shape of the DataFrame using the .shape attribute:



You should get the following output:



This DataFrame contains 2944 rows and 22 columns.

1. Display the first five rows of the DataFrame using the head() method:



You should get the following output:

Table

Description automatically generated

1. Find the number of duplicate rows using the duplicated() and sum() methods:

Graphical user interface, website

Description automatically generated

You should get the following output:



Looking at the 22 columns in this dataset, we can see that there are 5 duplicate rows.

1. Display the duplicate rows using the loc() and duplicated() methods:

A picture containing chart

Description automatically generated

You should get the following output:

Table

Description automatically generated

The following rows are duplicates: 2939 to 2943

1. Display the duplicate rows just like we did in *Step 5*, but with the keep='last' parameter instead:

Text

Description automatically generated with low confidence

You should get the following output:

Table

Description automatically generated

By using the keep='last' parameter, the following rows are considered duplicates: 13, 14, 2930, 2930, 2930, 2931, and 2932. By comparing this output to the one from the previous step, we can see that rows 2939 and 13 are identical.

1. Remove the duplicate rows using the drop\_duplicates() method along with the keep='first' parameter and save this into a new DataFrame called df\_unique:

A picture containing company name

Description automatically generated

1. Display the shape of df\_unique with the .shape attribute:



You should get the following output:



Now that we have removed the five duplicate records, 2939 rows remain. Now, the dataset only contains unique observations.

In this exercise, you learned how to identify and remove duplicate records from a real-world dataset.

## Converting Data Types

Another problem you may face in a project is incorrect data types being inferred for some columns. As we saw in Chapter 10, Analyzing a Dataset, the pandas package provides us with a very easy way to display the data type of each column using the .dtypes attribute. You may be wondering, when did pandas identify the type of each column? The types are detected when you load the dataset into a pandas DataFrame using methods such as read\_csv(), read\_excel(), and so on.

When you've done this, pandas will try its best to automatically find the best type according to the values contained in each column. Let's see how this works on the Cancer dataset.

Let’s print the data type of each column:

Graphical user interface, text, application

Description automatically generated

You should get the following output:

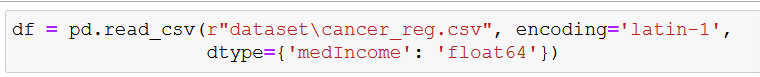
Table

Description automatically generated with medium confidence

The preceding output shows the data types that have been assigned to each column. popEst2015, MedianAge, and MedianAgeMale have been identified as numerical variables (int64, float64), Country and City are object variable, This is not too bad. pandas did a great job of recognizing non-text columns.

But what if you want to change the types of some columns? You have two ways to achieve this.

The first way is to reload the dataset, but this time, you will need to specify the data types of the columns of interest using the dtype parameter. This parameter takes a dictionary with the column names as keys and the correct data types as values, Let's try this on medIncome. We know this is a numerical variable as it contains a integers (code). Here, we are going to change its type to float:



Let’s check the data types with .dtypes

Graphical user interface, text

Description automatically generated

You should get the following output:

Table

Description automatically generated with medium confidence

As you can see, the data type for medIncome has effectively changed to a float64 type.

Now, let's look at the second way of converting a single column into a different type. In pandas, you can use the astype() method and specify the new data type that it will be converted into as its parameter. It will return a new column (a new pandas series, to be more precise), so you need to reassign it to the same column of the DataFrame. For instance, if you want to change the medInome column back to a int64 variable, you would do the following:

Graphical user interface, application

Description automatically generated with medium confidence

You should get the following output:

A picture containing table

Description automatically generated

You can see, the data type of medIncome has changed back to int64.

Now, Let’s try changing the data types of Country variable from object to category. You can achieve by the following code:

Chart

Description automatically generated with low confidence

You should get the following output:

Table

Description automatically generated with medium confidence

As you can see, the data type for Country has changed to a categorical variable. The difference between object and category is that the latter has a finite number of possible values (also called discrete variables). Once these have been changed into categorical variables, pandas will automatically list all the values. They can be accessed using the .cat.categories attribute:

Text

Description automatically generated with low confidence

You should get the following output:

Text

Description automatically generated

pandas has identified that there are 1819 different values in this column and has listed all of them. Depending on the data type that's assigned to a variable, pandas provides different attributes and methods that are very handy for data transformation or feature engineering (this will be covered in Chapter 12, Feature Engineering).

As a final note, you may be wondering when you would use the first way of changing the types of certain columns (while loading the dataset). To find out the current type of each variable, you must load the data first, so why will you need to reload the data again with new data types? It will be easier to change the type with the astype() method after the first load. There are a few reasons why you would use it. One reason could be that you have already explored the dataset on a different tool, such as Excel, and already know what the correct data types are.

The second reason could be that your dataset is big, and you cannot load it in its entirety. As you may have noticed, by default, pandas use 64-bit encoding for numerical variables. This requires a lot of memory and may be overkill.

## Exercise 11.02: Converting Data Types for the Life Expectancy Dataset

In this exercise, you will prepare a dataset by converting its variables into the correct data types.

We will be continuing with Life Expectancy dataset in this exercise.

This dataset can be found in this book's GitHub repository:

1. Import the pandas package and dataset.
2. Print the data type of each column using the dtypes attribute:

Graphical user interface, application

Description automatically generated with medium confidence

You should be getting the following output:

Table

Description automatically generated

1. Using the astype() method, convert the ‘Status' column into a categorical variable, as shown in the following code snippet:

A picture containing text

Description automatically generated

1. Convert the ‘Country’column into categorical variables, like we did in the previous step:

A picture containing diagram

Description automatically generated

1. Create a for loop that will iterate through the two categorical columns ('Status', and 'Country') and print their names and categories using the .cat.categories attribute:

Graphical user interface, text

Description automatically generated

You should be getting the following output:

Text

Description automatically generated

1. Create a new DataFrame called obj\_df that will only contain variables of the object type using the select\_dtypes method along with the include=’category’ parameter:

A picture containing text

Description automatically generated

1. Create a new variable called obj\_cols that contains a list of column names from the obj\_df DataFrame using the .columns attribute and display its content:

Graphical user interface

Description automatically generated with low confidence

You should be getting the following output:



1. Like we did in *Step 5*, create a for loop that will iterate through the column names contained in obj\_cols and print their names and unique values using the unique() method:

Graphical user interface, text, application

Description automatically generated

You should be getting the following output:

A screenshot of a computer

Description automatically generated with medium confidence

1. Now, create a for loop that will iterate through the column names contained in obj\_cols and convert each of them into a categorical variable using the astype() method:

A picture containing diagram

Description automatically generated

1. Print the data type of each column using the dtypes attribute:

Graphical user interface, text, application

Description automatically generated

You should be getting the following output:

Table

Description automatically generated with medium confidence

The preceding output has been truncated.

You have successfully converted the columns that have incorrect data types (numerical or object) into categorical variables. Your dataset is now one step closer to being prepared for modeling.

In the next section, we will look at handling incorrect values.

## Handling Missing Values

So far, you have looked at a variety of issues when it comes to datasets. Now it is time to discuss another issue that occurs quite frequently: missing values. As you may have guessed, this type of issue means that certain values are missing for certain variables.

The pandas package provides a method that we can use to identify missing values in a DataFrame: .isna(). Let's see it in action on the Cancer dataset. First, you need to import pandas and load the data into a DataFrame:

Text

Description automatically generated

You should be getting the following output:

Table

Description automatically generated

As we saw previously, we can give the output of a binary variable to the .sum() method, which will add all the True values together (cells that have missing values) and provide a summary for each column:



You should be getting the following output:

Table

Description automatically generated

As you can see, there are 2291 missing values in the 'PctSomeCol18\_24' column and 152 in the 'PctEmployed16\_Over'column. Let's have a look at the missing value observations for 'PctSomeCol18\_24'. You can use the output of the .isna() method to subset the rows with missing values:



You should be getting the following output:

Table

Description automatically generated

The pandas package provides a method that we can use to easily remove missing values: .dropna(). This method returns a new DataFrame without all the rows that have missing values. By default, it will look at all the columns. You can specify a list of columns for it to look for with the subset parameter.

In a real project, you will have to discuss these cases with the business and check whether these transactions are genuine or not. If the business confirms that these observations are irrelevant, then you will need to remove them from the dataset.

A picture containing icon

Description automatically generated

You should be getting the following output:

Table

Description automatically generated

This method returns a new DataFrame with no missing values for the specified columns. If you want to replace the original dataset directly, you can use the inplace=True parameter:



Now, look at the summary of the missing values for each variable:

Text

Description automatically generated

You should be getting the following output:

A picture containing table

Description automatically generated

We can impute the missing value with values that seem genuine. In most of the cases we will be using mean or median to fill the missing values. To demonstrate the how to impute I will using “missing” word to fill the null values.

In the exercise we will see how to impute the missing values with mean or median.



Let's see if we have any missing values in the dataset:

Graphical user interface, application, Teams

Description automatically generated

You should be getting the following output:

Table

Description automatically generated

Let's see if we have any missing values in the dataset:



You should be getting the following output:

Table

Description automatically generated

So, we left will one variable having the missing values. Let’s go ahead and drop them.



## Exercise 11.04: Fixing Missing Values in the Life Expectancy Dataset

In this exercise, you will be cleaning out all the missing values for all the numerical variables in the Life Expectancy dataset.

the dataset file that we'll be using in this exercise has been uploaded to this book's GitHub repository:

1. Import Pandas and dataset to continue with the exercise.
2. Print the first five rows of the DataFrame using the .head() method:

Graphical user interface, application

Description automatically generated with medium confidence

You should get the following output:

Table

Description automatically generated

1. Print the data type of each column using the dtypes attribute:

Graphical user interface, application

Description automatically generated

You should get the following output:

Text

Description automatically generated with medium confidence

1. Print the number of missing values for each column by combining the .isna() and .sum() methods:

Graphical user interface

Description automatically generated

You should get the following output:

Text

Description automatically generated with medium confidence

1. Create a condition mask called ‘LE\_mask’ so that you can find the missing values in the ‘Life expectancy’ column using the .isna() method:

Text

Description automatically generated

1. Display the number of missing values for this column by using the .sum() method on ‘LE\_mask’:



You should get the following output:



Here, you got the exact same number of missing values for ‘Life expectancy’ that you did in Step 4.

1. Extract the median of ‘Life expectancy’ using the .median() method and store it in a new variable called ‘LE\_median’. Print its value:

A picture containing text

Description automatically generated

You should get the following output:



The median value for this column is 72. You will replace all the missing values with this value in the ‘Life expectancy’ column.

1. Replace all the missing values in the ‘Life expectancy’ variable with their median using the .fillna() method, along with the inplace=True parameter:



1. Print the number of missing values for X0 by combining the .isna() and .sum() methods:

Text

Description automatically generated with low confidence

You should get the following output:



There are no more missing values in the variables.

1. Create a for loop that will iterate through all the columns of the DataFrame. In the for loop, calculate the median for each and save them into a variable called col\_median. Then, impute missing values with this median value using the .fillna() method, along with the inplace=True parameter, and print the name of the column and its median value:

Text

Description automatically generated

You should get the following output:

Graphical user interface

Description automatically generated with low confidence

1. Print the number of missing values for each column by combining the .isna() and .sum() methods:



You should get the following output:

Table

Description automatically generated

You have successfully fixed the missing values for all the numerical variables using the methods provided by the pandas package: .isna() and .fillna().

## Activity 11.01: Preparing the Bikes Dataset

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

You just received the dataset from your partner company but realized it is not as clean as you expected; there are missing and incorrect values. Your task is to fix the main data quality issues in this dataset.

The dataset to be used in this activity can be found on our GitHub repository:

https://github.com/fenago/DSBook/blob/main/Chapter%2010%20-%

The following steps will help you complete this activity:

* Download and load the dataset into Python using .read\_csv().
* Print out the dimensions of the DataFrame using .shape.
* Check for duplicate rows by using .duplicated() and .sum() on all the columns.
* Check for unexpected values for the following numerical variables
* Replace the identified incorrect values.
* Check the data type of the different columns using .dtypes.
* Change the data types to categorical for the columns that don't contain numerical values using .astype().
* Check for any missing values using .isna() and .sum() for each numerical variable.
* Replace the missing values for each numerical variable with their corresponding mean or median values using .fillna(), .mean(), and .median().

## Summary

In this chapter, you learned how important it is to prepare any given dataset and fix the main quality issues it has. This is critical because the cleaner a dataset is, the easier it will be for any machine learning model to easily learn about the relevant patterns. On top of this, most algorithms can't handle issues such as missing values, so they must be handled prior to the modeling phase. In this chapter, you covered the most frequent issues that are faced in data science projects: duplicate rows, incorrect data types, unexpected values, and missing values.

The goal of this chapter was to introduce you to the concepts that will help you to spot some of these issues and easily fix them so that you have the basic toolkit to be able to handle other cases. As a final note, throughout this chapter, we emphasized how important it is to discuss the issues you find with the business or the data engineering team you are working with. For instance, if you've detected unexpected values in a dataset, you may want to confirm that they don't have any special meaning from a business point of view before removing or replacing them.

You also need to be very careful when fixing issues: you don't want to alter the dataset too much so that it creates additional unexpected patterns. This is exactly why it is recommended that you replace any of the missing values of numerical variables with their mean or median. Otherwise, you will change its distribution drastically. For example, if the values of a variable are between 0 and 10, replacing all the missing values with -999 will drastically change their mean and standard deviation.

In the next chapter, we will discuss the interesting topic of feature engineering.