# CHAPTER 12: Feature Engineering

## Introduction

In the previous chapters, we learned how to analyze and prepare a dataset in order to increase its level of quality. In this chapter, we will introduce you to another interesting topic: creating new features, also known as feature engineering. You already saw some of these concepts in Chapter 3, Binary Classification, but we will dive a bit deeper into it in this chapter.

The objective of feature engineering is to provide more information for the analysis you are performing on or the machine learning algorithms you will train on. Adding more information will help you to achieve better and more accurate results.

New features can come from internal data sources such as another table from databases or from different systems. For instance, you may want to link data from the CRM tool used in your company to the data from a marketing tool. The added features can also come from external sources such as open-source data or shared data from partners or providers. For example, you may want to link the volume of sales with a weather API or with governmental census data. But it can also come from the original dataset by creating new variables from existing ones.

Let's pause for a second and understand why feature engineering is so important for training machine learning algorithms. We are all aware that these algorithms have achieved incredible results in recent years in finding extremely complex patterns from data. But their main limitations lie in the fact that they can only analyze and find meaningful patterns within the data provided as input. If the data is incorrect, incomplete, or missing important features, the algorithms will not be able to perform correctly.

On the other hand, we humans tend to understand the broader context and see the bigger picture quite easily. For instance, if you were tasked with analyzing customer churn, even before looking at the existing data, you would already expect it to have some features describing customer attributes such as demographics, services or products subscribed to, and subscription date. And once we receive the data, we can highlight the features that we think are important and missing from the dataset. This is the reason why data scientists, with their expertise and experience, need to think about the additional information that will help algorithms to understand and detect more meaningful patterns from this enriched data. Without further ado, let's jump in.

## Merging Datasets

Most organizations store their data in data stores such as databases, data warehouses, or data lakes. The flow of information can come from different systems or tools. Most of the time, the data is stored in a relational database composed of multiple tables rather than a single one with well-defined relationships between them.

For instance, an online store could have multiple tables for recording all the purchases made on its platform. One table might contain information relating to existing customers, another one might list all existing and past products in the catalog, and a third one might contain all of the transactions that occurred, and so on.

If you were working on a project recommending products to customers for an e-commerce platform such as Amazon, you may have been given only the data from the transactions table. In that case, you would like to get some attributes for each product and customer and would have to ask to extract these additional tables you need and then merge the three tables together before building your recommendation system.

Let's see how we can merge multiple data sources with a real example: I have taken some real data from the KillBiller Application and some downloaded data, contained in three csv files:

1. [user\_usage.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/user_usage.csv) – A first dataset containing users monthly mobile usage statistics
2. [user\_device.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/user_device.csv) – A second dataset containing details of an individual “use” of the system, with dates and device information.
3. [android\_devices.csv](https://github.com/shanealynn/Pandas-Merge-Tutorial/blob/master/android_devices.csv) – A third dataset with device and manufacturer data, which lists all Android devices and their model code, obtained from Google

You can download the datasets from the below github link:

We can load these CSV files as [Pandas DataFrames](http://pandas.pydata.org/pandas-docs/stable/dsintro.html?highlight=dataframe) into pandas using the Pandas [read\_csv](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) command, and examine the contents using the [DataFrame head()](http://pandas.pydata.org/pandas-docs/stable/basics.html#basics) command.

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

Let’s check the shape

Graphical user interface, text, application

Description automatically generated

Sample usage information from the KillBiller application showing monthly mobile usage statistics for a subset of users.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Let’s check the shape

Graphical user interface, application, Word

Description automatically generated

User information from KillBiller application giving the device and OS version for individual “uses” of the KillBiller application.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Let’s check the shape

Graphical user interface, text, application, Word

Description automatically generated

Android Device data, containing all Android devices with manufacturer and model details.

There are linking attributes between the sample datasets that are important to note – “use\_id” is shared between the user\_usage and user\_device, and the “device” column of user\_device and “Model” column of the devices dataset contain common codes.

We would like to determine if the usage patterns for users differ between different devices. For example, do users using [Samsung](http://www.samsung.com/ie/) devices use more call minutes than those using  [LG](http://www.lg.com/uk) devices? This is a toy problem given the small sample size in these dataset, but is a perfect example of where merges are required.

We want to form a single dataframe with columns for user usage figures (calls per month, sms per month etc) and also columns with device information (model, manufacturer, etc). We will need to “merge” (or “join”) our sample datasets together into one single dataset for analysis.

There are multiple ways to join two tables together:

Graphical user interface, text, application

Description automatically generated

**The Left Join**

The left join will keep all the rows from the first DataFrame, which is the *user\_usage* dataset (the left-hand side) and join it to the matching rows from the second DataFrame, which is the *user\_device* dataset (the right-hand side), as shown in *Figure*

Diagram, venn diagram

Description automatically generated

To perform a left join, we need to specify to the .merge() method.

A picture containing company name

Description automatically generated

The result dataframe contains the successfully matched items on the top, and at the bottom contains the rows in user\_usage that didn’t have a corresponding use\_id in user\_device. We can validate this with the help of shape method.

Text

Description automatically generated

Let’s also see top and bottom 5 records in result dataframe.



Table

Description automatically generated

A picture containing text

Description automatically generated

Table

Description automatically generated

Rows in the left dataframe that have no corresponding join value in the right dataframe are left with NaN values.

**The Right Join**

The right join is similar to the left join except it will keep all the rows from the second DataFrame (the right-hand side) and tries to match it with the first one (the left-hand side), as shown in *Figure*

Diagram, venn diagram

Description automatically generated

For examples sake, we can repeat this process with a right join / right merge, simply by replacing *how=’left’* with *how=’right’* in the Pandas merge command.

A picture containing chart

Description automatically generated

The result expected will have the same number of rows as the right dataframe, user\_device, but have several empty, or NaN values in the columns originating in the left dataframe, user\_usage (namely “outgoing\_mins\_per\_month”, “outgoing\_sms\_per\_month”, and “monthly\_mb”). Conversely, we expect no missing values in the columns originating in the right dataframe, “user\_device”.

Text

Description automatically generated

Note that the output has the same number of rows as the right dataframe, with missing values only where use\_id in the left dataframe didn’t match anything in the left.

There are two other types of merging: inner and outer.

An inner join will only keep the rows that match between the two tables:

Diagram, venn diagram

Description automatically generated

You just need to specify the how = 'inner' parameter in the .merge() method.

A picture containing text

Description automatically generated

Lets check the shape of the result dataframe

Text

Description automatically generated

We can see there are no observations common in both the dataframe based on use\_id.

The outer join will keep all rows from both tables (matched and unmatched), as shown in *Figure*

Diagram, venn diagram

Description automatically generated

As you may have guessed, you just need to specify the how == 'outer' parameter in the .merge() method:

Company name

Description automatically generated with low confidence

Let’s check the shape of the result dataframe.

Text

Description automatically generated

Before merging two tables, it is extremely important for you to know what your focus is. If your objective is to expand the number of features from an original dataset by adding the columns from another one, then you will probably use a left or right join. But be aware you may end up with more observations due to potentially multiple matches between the two tables. On the other hand, if you are interested in knowing which observations matched or didn't match between the two tables, you will either use an inner or outer join.

## Exercise 12.01: Merging the Marketing Dataset with the Revenue Data

In this exercise, we will merge the marketing dataset with revenue dataset.

You can download the dataset from the following github link:

We begin with importing the pandas package.

Graphical user interface, text, application

Description automatically generated

Using the .read\_csv() method from the pandas package, load the marketing dataset into a new DataFrame called df\_marketing:



Display the dimensions of this DataFrame using the .shape attribute:

Graphical user interface, text, application

Description automatically generated

The Marketing dataset contain 124 rows and 3 columns.

A picture containing chart

Description automatically generated

Table

Description automatically generated

Using the .read\_csv() method from the pandas package, load the revenue raw dataset into a new DataFrame called df\_revenue:



Display the dimensions of this DataFrame using the .shape attribute:

Graphical user interface, application, Word

Description automatically generated

The Revenue dataset contain 119 rows and 4 columns.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Both DataFrames have a column called Date, so we will use it to merge them together.

We are interested in learning more about each of these Date. Let's make sure they are all unique in this dataset.

Graphical user interface, application

Description automatically generated

There are 124 unique date values in marketing dataframe.

Now, lets see in the revenue dataframe.

Graphical user interface, text, application

Description automatically generated

There are 119 unique date values in marketing dataframe.

This corresponds exactly to the number of rows of this DataFrame, so we're absolutely sure this column contains unique values. This also means that after merging the two tables, there will be only one-to-one matches. We won't have a case where we get multiple rows from one of the datasets matching with only one row of the other one.

Perform a left join on the two DataFrames using the .merge() method and save the results into a new DataFrame called merged\_df. Specify the how='left' and on=’Date’ parameters:

Graphical user interface, text

Description automatically generated

Print the dimensions of the new merged DataFrame using the .shape attribute:

Graphical user interface, text, application

Description automatically generated

We got exactly 124 rows after merging, which is what we expect as we used a left join and there was a one-to-one match on the Date column from both original DataFrames. Also, we now have 6 columns, which is the objective of this exercise. But before concluding it, we want to see whether there are any Date that didn't match between the two datasets. To do so, we will be looking at one column from the right-hand side DataFrame (the Revenue dataset) and see if there are any missing values.

Print the total number of missing values from the 'Date' column by combining the .isna() and .sum() methods:





There are 56 Dates from the Marketing dataset that didn't match the Revenue dateset  Date variable.

Print the missing postcodes using the .iloc() method, as shown in the following code snippet:

A picture containing graphical user interface

Description automatically generated

You should get the following output:

Table

Description automatically generated

Table

Description automatically generated with medium confidence

In a real project, you would have to get in touch with your stakeholders or the data team to see if you are able to get this data.

We have successfully merged the two datasets of interest and have expanded the number of features from 3 to 6. We now have a much richer dataset and will be able to perform a more detailed analysis of it.

## Binning Variables

As mentioned earlier, feature engineering is not only about getting information not present in a dataset. Quite often, you will have to create new features from existing ones. One example of this is consolidating values from an existing column to a new list of values.

For instance, you may have a very high number of unique values for some of the categorical columns in your dataset, let's say over 1,000 values for each variable. This is actually quite a lot of information that will require extra computation power for an algorithm to process and learn the patterns from. This can have a significant impact on the project cost if you are using cloud computing services or on the delivery time of the project.

One possible solution is to not use these columns and drop them, but in that case, you may lose some very important and critical information for the business. Another solution is to create a more consolidated version of these columns by reducing the number of unique values to a smaller number, let's say 100. This would drastically speed up the training process for the algorithm without losing too much information. This kind of transformation is called binning and, traditionally, it refers to numerical variables, but the same logic can be applied to categorical variables as well.

In the following examples, we’ll be exploring and engineering features from a dataset with information about voter demographics and participation. I’ve selected 3 categorical variables to work with:

You can download the dataset from the following link:

**Importing the required packages**

Text, table

Description automatically generated

We can load these CSV files as [Pandas DataFrames](http://pandas.pydata.org/pandas-docs/stable/dsintro.html?highlight=dataframe) into pandas using the Pandas [read\_csv](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) command, and examine the contents using the [DataFrame head()](http://pandas.pydata.org/pandas-docs/stable/basics.html#basics) command.



Let’s check the top five observations

Table

Description automatically generated

Now, we can see the unique variables and number of unique variables using the following lines of code.

Graphical user interface

Description automatically generated with low confidence

Graphical user interface, text, application, Word

Description automatically generated

We have 6 unique labels in the ‘party\_cd’ variable.

First, let’s check out ‘party\_cd’. The image below show how many individual voters belong to each political party.



Chart, bar chart

Description automatically generated

There are so few registered Libertarians, Constitutionalists, and members of the Green Party that we can barely see them on the graph. These would be good examples of rare labels. For the purpose of this chapter, we’ll define rare label as those that make up less than 5% observations. This is a common threshold for defining rare labels, but ultimately that’s up to your discretion.

Let’s look at a breakdown of the actual numbers:

Text

Description automatically generated

Those three categories each make up far less than 5% of the population. Even if we lumped them all together into a single category, that new category would still represent less than 1% voters.

“REP” and “DEM” represent the two major political parties, whereas “UNA” represents voters that registered as unaffiliated with a political party. So here, it could make sense to lump in our three rare labels into that unaffiliated group so that we have three categories: one for each of the two major parties, and a third representing individuals that chose not to align with either major party.

First, let’s create a new column called ‘party\_bin’ by copying the ‘party\_cd’ column:



Then, we are going to create a list called ‘Other\_voters’ containing the names of Libertarians, Constitutionalists, and members of the Green Party.



And finally, using .loc() and .isin() methods from pandas, we are going to change the value of party\_bin to Others for all of the voters who are present in the Other\_voters list:



Now, if we print the list of unique values for this new column, we will see the minority voters (LIB, CST, and GRE) have been replaced by the value “Others”.

Graphical user interface

Description automatically generated with low confidence

A picture containing graphical user interface

Description automatically generated

4 is the number of unique values of ‘party\_bin’ column. So we reduced the number of unique values in this column from 6 to 4. We just saw how to group categorical values together, but the same process can be applied to numerical values as well. For instance, it is quite common to group people's ages into bins such as 20s (20 to 29 years old), 30s (30 to 39), and so on.

Have a look at Exercise 12.02, Binning the “Day\_Name” variable from the Revenue dataset.

## Exercise 12.02: Binning the Day\_Name Variable from the Revenue Dataset

The dataset we will be using in this exercise is the Ames Housing dataset and it can be found in our GitHub repository:

Import the required packages

Graphical user interface, text

Description automatically generated with medium confidence

Loading the dataset and looking at the top 5 observations





Table

Description automatically generated

We are going to deal with Day\_Name and Visitors Columns.

Let’s check the number of unique values from the Day\_Name variable.

Graphical user interface

Description automatically generated with low confidence

We can see the unique values from the Day\_Name column using the following code.

Graphical user interface, text, application

Description automatically generated

We want to create the weekend bin by combining the “Saturday” and “Sunday” labels. Before that lets visualize the number of visitors in each day of the week.

A picture containing graphical user interface

Description automatically generated

Chart, bar chart

Description automatically generated

Create a copy of ‘Day\_Name’ column and naming the variable with “Weekend\_bin”.



Create a list called “Weekend” containing the names “Saturday” and “Sunday”



Finally, change the values of “Weekend\_bin” for the day that are present in Weekend\_list.



Print the list of unique values for this this new column, to see the Saturday and Sunday have been replaced by the value Weekend.

Graphical user interface, text, application

Description automatically generated

Check the number of unique values in the “Weekend\_bin”.

Graphical user interface, application, Word

Description automatically generated

## Manipulating Dates

In most datasets you will be working on, there will be one or more columns containing date information. Usually, you will not feed that type of information directly as input to a machine learning algorithm. The reason is you don't want it to learn extremely specific patterns, such as customer A bought product X on August 3, 2012, at 08:11 a.m. The model would be overfitting in that case and wouldn't be able to generalize to future data.

What you really want is the model to learn patterns, such as customers with young kids tending to buy unicorn toys in December, for instance. Rather than providing the raw dates, you want to extract some cyclical characteristics such as the month of the year, the day of the week, and so on. We will see in this section how easy it is to get this kind of information using the pandas package.

Note

There is an exception to this rule of thumb. If you are performing a time-series analysis, this kind of algorithm requires a date column as an input feature, but this is out of the scope of this book.

In *Chapter 10*, *Analyzing a Dataset* you were introduced to the concept of data types in pandas. At that time, we mainly focused on numerical variables and categorical ones but there is another important one: datetime.

We will be working on catalog dataset; you can find the dataset from the below github link:

Import the required packages and dataset.



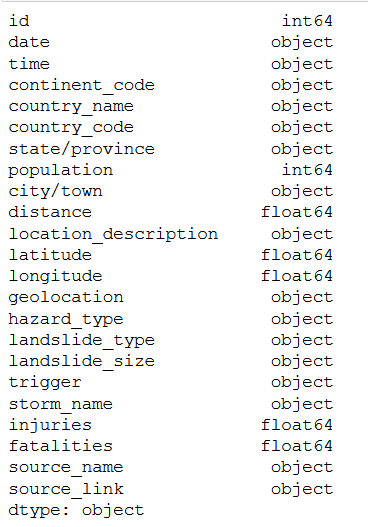


Let’s look at the data type of each column.

Graphical user interface

Description automatically generated with low confidence

You should get the following output:



We can see that pandas did not automatically detected that “date” variable is of type datetime. But for some other datasets, it may recognize dates properly. In this case, you will have to manually convert them using the .to\_datetime() method:

Text

Description automatically generated with low confidence

Once the column is converted to datetime, pandas provides a lot of attributes and methods for extracting time-related information. For instance, if you want to get the year of a date, you use the .dt.year attribute:



You should get the following output:

Table

Description automatically generated

As you may have guessed, there are attributes for extracting the month and day of a date: .dt.month and .dt.day respectively. You can get the day of the week from a date using the .dt.dayofweek attribute:

A picture containing text

Description automatically generated

You should get the following output:

Table

Description automatically generated

With datetime columns, you can also perform some mathematical operations. We can, for instance, add 3 days to each date by using pandas time-series offset object, pd.tseries.offsets.Day(3):

A picture containing text

Description automatically generated

You should get the following output:

Text, table

Description automatically generated

You can also offset days by business days using pd.tseries.offsets.BusinessDay(). For instance, if we want to get the previous business days, we do:

A picture containing text

Description automatically generated

You should get the following output:

Table

Description automatically generated

Another interesting date manipulation operation is to apply a specific time-frequency using pd.Timedelta(). For instance, if you want to get the first day of the month from a date, you do:

A picture containing text

Description automatically generated

You should get the following output:

Text

Description automatically generated

As you have seen in this section, the pandas package provides a lot of different APIs for manipulating dates. You have learned how to use a few of the most popular ones. You can now explore the other ones on your own.

## Exercise 12.03: Date Manipulation on Revenue dataset

1. Import the pandas package:



1. Use the .read\_csv() method from the pandas package and load the dataset into a new DataFrame called df:

A picture containing text

Description automatically generated

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

Table

Description automatically generated

1. Print out the data types for each column using the .dtypes attribute:

Graphical user interface, text, application

Description automatically generated

1. Convert the “Date” to datetime using the pd.to\_datetime() method:

Text

Description automatically generated with low confidence

1. Print out the data types for each column using the .dtypes attribute:



You should get the following output:

Graphical user interface, text

Description automatically generated with medium confidence

1. Create a new column called Year, which will contain the year of each date from the Date column using the .dt.year attribute:

A picture containing chart

Description automatically generated

1. Create a new column called Month, which will contain the month of each date using the .dt.month attribute:

A picture containing text

Description automatically generated

1. Create a new column called Day, which will contain the day of the month for each date using the .dt.day attribute:

Chart

Description automatically generated with medium confidence

1. Create a new column called Dow, which will contain the day of the week for each date using the .dt.dayofweek attribute:

Chart

Description automatically generated with low confidence

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

Text

Description automatically generated

You should get the following output:

Table

Description automatically generated

1. We can see we have successfully created four new features: Year, Month, Day, and Dow. Now let's create another that will indicate whether the date was during a weekend or not.

A picture containing chart

Description automatically generated

1. Create a new column called IsWeekend, which will contain binary values indicating whether the Dow column is over or equal to 5 (0 corresponds to Monday, 5 and 6 correspond to Saturday and Sunday respectively):



1. Display the first 5 rows using the .tail() method:

Text

Description automatically generated

You should get the following output:

Table

Description automatically generated

## Performing Data Aggregation

Alright. We are getting close to the end of this chapter. But before we wrap it up, there is one more technique to explore for creating new features: data aggregation. The idea behind it is to summarize a numerical column for specific groups from another column.

You may wonder to yourself in which cases you would want to perform feature engineering using data aggregation. If you already have a numerical column that contains a value for each record, why would you need to summarize it and add this information back to the DataFrame? It feels like we are just adding the same information but with fewer details. But there are actually multiple good reasons for using this technique.

One potential reason might be that you want to normalize another numerical column using this aggregation. For instance, if you are working on a dataset for a retailer that contains all the sales for each store around the world, the volume of sales may differ drastically for a country compared to another one as they don't have the same population. In this case, rather than using the raw sales figures for each store, you would calculate a ratio (or a percentage) of the sales of a store divided by the total volume of sales in its country. With this new ratio feature, some of the stores that looked as though they were underperforming because their raw volume of sales was not as high as for other countries may actually be performing much better than the average in its country.

In pandas, it is quite easy to perform data aggregation. We just need to combine the following methods successively: .groupby() and .agg().

We will need to specify the list of columns that will be grouped together to the .groupby() method. If you are familiar with pivot tables in Excel, this corresponds to the Rows field.

The .agg() method expects a dictionary with the name of a column as a key and the aggregation function as a value such as {'column\_name': 'aggregation\_function'}. In an Excel pivot table, the aggregated column is referred to as values.

Let’s see how to do it on the catalog dataset. First, we need to import the data:





Let’s calculate the total population for each country. We will specify the “country\_name” column as the grouping column:



You should get the following output:

Table

Description automatically generated

This result gives the total population of each country. This level of information may be too high-level and we may want a bit more granular detail. Let's perform the same aggregation but this time we will group on two columns: Country and State/province. We just need to provide the names of these columns as a list to the .groupby() method:



You should get the following output:

Graphical user interface, application

Description automatically generated

We can add one more layer of information and get the population for each country, state/province and city/town. We can add this new column in the .groupby() method:

A picture containing text

Description automatically generated

You should get the following output:

Table

Description automatically generated

We have generated a new DataFrame with total population per county, state/province and city/town. We can now merge this additional information back into the original DataFrame. But before that, there is an additional data transformation step required: reset the column index. The pandas package creates a multi-level index after data aggregation by default. You can think of it as though the column names were stored in multiple rows instead of one only. To change it back to a single level, you need to call the .reset\_index() method:



Let’s see the top 5 observations

Graphical user interface

Description automatically generated with medium confidence

You should get the following output:

Table

Description automatically generated

## Exercise 12.04: Feature Engineering Using Data Aggregation on Bigmart sales dataset

In this exercise, we will create new features using data aggregation. First, we’ll maximum ‘Item\_outlet\_sales’ based on “outlet\_size” and “outlet\_establishmen\_year” and we will add this information back to the dataset, and then, we will calculate the ratio of those two. And finally, we will groupby with multiple indexes to get the better insights from our data.

You can download the dataset used in this exercise from the following github link:

Import the required packages

Graphical user interface, text, table

Description automatically generated

Import the dataset and check the top 5 observations

Word

Description automatically generated with low confidence

You should get the following output:

Table

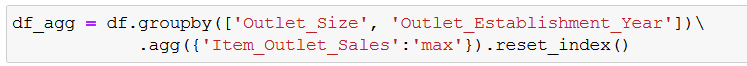
Description automatically generated

Drop the null values if any

Text

Description automatically generated with low confidence

Calculate the maximum “item\_outlet\_sales” by grouping the data based on “outlet\_size” and “outlet\_establishment\_year”, then assign the result to the new dataframe.



Print the top 5 observations using the following command



You should get the following output:

Table

Description automatically generated

Rename the column name

A picture containing text

Description automatically generated



You should get the following output:

Table

Description automatically generated

Now, merge the old data frame with the aggregated data frame on ‘outlet\_size’ and ‘outlet\_establishment\_year’ and assign the results to the df\_new.

A picture containing graphical user interface

Description automatically generated

Create a new column called SalesRatio by dividing “Item\_Outlet\_Sales” with “Item\_Outlet\_Sales\_Max”.

A picture containing text

Description automatically generated

Print out the first five rows of df\_new:

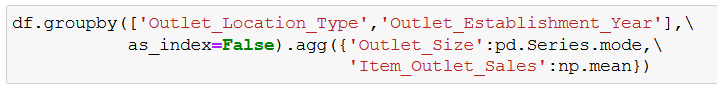


You should get the following output:

Table

Description automatically generated

Groupby the data on ‘Outlet\_Location\_Type” and “Outlet\_Establishment\_Year”. Then check most number of outlets based on size and finally, calculate mean Item\_Outlet\_Sales based on size of the outlet



You should get the following output:

Table

Description automatically generated

Calculate the mean\_MRP and mean\_Sales of iteam\_type by grouping the data on Outlet\_Type.

Text

Description automatically generated

You should get the following output:

Graphical user interface, table

Description automatically generated

## Activity 12.01: Feature Engineering on a Financial Dataset

You are working for a major bank in the Czech Republic and you have been tasked to analyze the transactions of existing customers. The data team has extracted all the tables from their database they think will be useful for you to analyze the dataset. You will need to consolidate the data from those tables into a single DataFrame and create new features in order to get an enriched dataset from which you will be able to perform an in-depth analysis of customers' banking transactions.

The datasets you will be using in this activity can be found on our Github repository:

You will be using only the following four tables:

Graphical user interface, text, application

Description automatically generated

The following steps will help you complete this activity:

1. Download and load the different tables from this dataset into Python.
2. Analyze each table with the .shape and .head() methods.
3. Find the common/similar column(s) between tables that will be used for merging based on the analysis from Step 2.
4. There should be four common tables. Merge the four tables together using pd.merge().
5. Rename the column names after merging with .rename().
6. Check there is no duplication after merging with .duplicated() and .sum().
7. Transform the data type for date columns using .to\_datetime().
8. Create two separate features from birth\_number to get the date of birth and sex for each customer.

Note

This is the rule used for coding the data related to birthday and sex in this column: the number is in the YYMMDD format for men, the number is in the YYMM+50DD format for women, where YYMMDD is the date of birth.

1. Fix data quality issues with .isna().
2. Create a new feature that will calculate customers' ages when they opened an account using date operations

## Summary

We first learned how to analyze a dataset and get a very good understanding of its data using data summarization and data visualization. This is very useful for finding out what the limitations of a dataset are and identifying data quality issues. We saw how to handle and fix some of the most frequent issues (duplicate rows, type conversion, value replacement, and missing values) using pandas' APIs.

Finally, we went through several feature engineering techniques. It was not possible to cover all the existing techniques for creating features. The objective of this chapter was to introduce you to critical steps that can significantly improve the quality of your analysis and the performance of your model. But remember to regularly get in touch with either the business or the data engineering team to get confirmation before transforming data too drastically. Preparing a dataset does not always mean having the cleanest dataset possible but rather getting the one that is closest to the true information the business is interested in. Otherwise, you may find incorrect or meaningless patterns. As we say, *with great power comes great responsibility*.

The next chapter opens a new part of this book that presents data science use cases end to end. *Chapter 13*, *Imbalanced Datasets*, will walk you through an example of an imbalanced dataset and how to deal with such a situation.