INTRODUCTION

In the previous chapter, we looked at various techniques to do with probability theory and hypothesis testing, which are often applied in data analysis. In this chapter, we will extend our knowledge by introducing mathematical models that are suitable for both data analysis and predictions. In this way, we will obtain the fundamental tools for deriving explanatory models and provide a generic framework for identifying causalities and effects when performing data analysis.

Direct marketing campaigns are a classical approach to increasing business revenue, informing potential customers about new products, and merchandising them. Having targeted marketing campaigns can significantly increase success rates and revenue since the audience is based on precise criteria and the analysis of past marketing campaigns. Thus, extracting information about successful campaigns and customers can significantly reduce marketing costs and increase sales.

In this chapter, we will analyze data from the direct marketing campaign of a Portuguese banking institution based on phone calls that were performed between May 2008 and November 2010. We will not only use the techniques presented in the previous chapters but will introduce new concepts, such as linear and logistic regression, that are widely used in data analysis and predictive modeling (predictive modeling being the process of using data patterns to predict future outcomes). These types of models have several advantages, but two of the most important ones are their simplicity and interpretability.

> **NOTE**
>
> The original dataset can be found here: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.
>
> You can also find it in our GitHub repository at: https://packt.live/2UOmIyp.
>
> For further reading on this topic, refer to the following article: [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. *A Data-Driven Approach to Predict the Success of Bank Telemarketing*. Decision Support Systems, Elsevier, 62:22-31, June 2014.

The scope of the marketing campaign is selling long-term deposits. For each call, information about the client and the outcome of the call is registered. For instance, client information contains family and education status and the client's current financial situation (the data is anonymized). Information about the last call from the previous marketing campaign is also registered. In this way, we have a clear picture of the type of client that has been contacted, as well as information about the historical outcome from previous marketing campaigns.

INITIAL DATA ANALYSIS

We'll start our analysis by loading the data into Python and performing some simple analysis, which will give us a feeling about the type of data and the different features of the dataset (this is presented in detail in *Figure 3.2*):

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

# pull data from github
bank_data = pd.read_csv("https://raw.githubusercontent.com/"\
                        "PacktWorkshops/"\
                        "The-Data-Analysis-Workshop/"\
                        "master/Chapter03/data/bank-additional/"\
                        "bank-additional-full.csv", sep=";")

# visualize the head of the dataset
bank_data.head().T
```

In the following screenshot, we can see the values that were returned by the previous command:

	0	1	2	3	4
age	56	57	37	40	56
job	housemaid	services	services	admin.	services
marital	married	married	married	married	married
education	basic.4y	high.school	high.school	basic.6y	high.school
default	no	unknown	no	no	no
housing	no	no	yes	no	no
loan	no	no	no	no	yes
contact	telephone	telephone	telephone	telephone	telephone
month	may	may	may	may	may
day_of_week	mon	mon	mon	mon	mon
duration	261	149	226	151	307
campaign	1	1	1	1	1
pdays	999	999	999	999	999
previous	0	0	0	0	0
poutcome	nonexistent	nonexistent	nonexistent	nonexistent	nonexistent
emp.var.rate	1.1	1.1	1.1	1.1	1.1
cons.price.idx	93.994	93.994	93.994	93.994	93.994
cons.conf.idx	-36.4	-36.4	-36.4	-36.4	-36.4
euribor3m	4.857	4.857	4.857	4.857	4.857
nr.employed	5191	5191	5191	5191	5191
y	no	no	no	no	no

Figure 3.1: Snapshot of the banking data, returned by the head().T function (first five entries)

Note that in this chapter, we are using the data provided in the **`bank-additional-full.csv`** file as a dataset. It contains slightly more features than the bank data itself (that is, **`bank-full.csv`**) as we believe that we will be able to perform more extensive analysis on the dataset by taking advantage of the additional information available.

A list of the available features of the dataset and their values can be found in the following table:

Column name	Description	Values
Age	Age of the contacted client	Numerical
Job	Type of job	admin., blue- collar, entrepreneur, housemaid, management, retired, self-employed, services, student, technician, unemployed, unknown
Marital	Marital status (note: "divorced" means divorced or widowed)	divorced, married, single, unknown
Education	Education level of the contacted client	basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university. degree, unknown
Default	Does the client have credit in default?	yes, no, unknown
Housing	Does the client have a housing loan?	yes, no, unknown
Loan	Does the client have a personal loan?	yes, no, unknown
Contact	Type of communication with the client	cellular, telephone
Month	Last contact, month of the year	jan, feb, …, nov, dec
day_of_week	Last contact, day of the week	mon, tue, wed, thu, fri
Duration	Last contact duration (in seconds)	Numeric
Campaign	Number of contacts performed during this campaign for this client	Numeric
Pdays	Number of days passed by after the client was contacted from a previous campaign	numeric (999 means the client was notcontacted)
Previous	Number of contacts performed before this campaign and for this client	Numeric
Poutcome	Outcome of the previous marketing campaign	failure, nonexistent, success
emp.var.rate	Employment variation rate (quarterly indicator)	Numeric
cons.price.idx	Consumer price index (monthly indicator)	Numeric
cons.conf.idx	Consumer confidence index (monthly indicator)	Numeric
euribor3m	Euribor 3-month rate	Numeric
nr.employed	Number of employees (quarterly indicator)	Numeric
Y	Has the client subscribed to a term deposit (outcome of the marketing campaign)?	yes, no

Figure 3.2: List of columns and their values, provided in the bank-additional-full.csv file

As shown in the preceding table, we have two types of columns in the dataset: numerical and categorical. A good practice in data analysis is to start by looking at the different features, their distributions, and possible outliers. We will do this in the following exercises.

EXERCISE 3.01: ANALYZING DISTRIBUTIONS OF NUMERICAL FEATURES IN THE BANKING DATASET

In this exercise, you will perform a simple analysis of the numerical features in the banking dataset. It is important to always start data analysis on a new dataset by deriving basic statistics. In this way, you will obtain general knowledge and a "feel" for the data. A typical example could be computing the minimum and maximum values in the **`age`** column. If those values are not aligned with your expectations (for example, a minimum age of 18, and a maximum in the range 70-90), you should investigate further to determine the reason for the disparity.

You will start by selecting the relevant numerical features in a programmatic way, computing some basic statistics (such as mean, standard deviation, and minimum and maximum values) on them, and plotting their distributions. This will give you an idea of the numerical features in the dataset. Follow these steps to complete this exercise:

1. First, select the relevant numerical features. A straightforward approach would be to just list them, but as in certain cases, the dimension of the data might be quite large (with tens or even hundreds of features), so you need a scalable approach that is independent of the size of the data. For this reason, use the **`np.issubdtype()`** function, which checks whether the type of the first argument is equal or a subtype of the second one:

```
# define numerical features
numerical_features = [col for col in bank_data.columns \
                      if np.issubdtype(bank_data[col]\
                      .dtype, np.number)]
print(numerical_features)
```

   The following is the output of the preceding code:

```
['age', 'duration', 'campaign', 'pdays', 'previous',
 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
 'euribor3m', 'nr.employed']
```

The detected numerical features are in line with those defined in *Figure 3.2*.

2. Next, compute some basic statistics on the identified numerical features (such as mean, standard deviation, minimum and maximum values, and quartiles):

```
# print statistics about the different numerical columns
bank_data[numerical_features].describe().T
```

The output will be as follows:

	count	mean	std	min	25%	50%	75%	max
age	41188.0	40.024060	10.421250	17.000	32.000	38.000	47.000	98.000
duration	41188.0	258.285010	259.279249	0.000	102.000	180.000	319.000	4918.000
campaign	41188.0	2.567593	2.770014	1.000	1.000	2.000	3.000	56.000
pdays	41188.0	962.475454	186.910907	0.000	999.000	999.000	999.000	999.000
previous	41188.0	0.172963	0.494901	0.000	0.000	0.000	0.000	7.000
emp.var.rate	41188.0	0.081886	1.570960	-3.400	-1.800	1.100	1.400	1.400
cons.price.idx	41188.0	93.575664	0.578840	92.201	93.075	93.749	93.994	94.767
cons.conf.idx	41188.0	-40.502600	4.628198	-50.800	-42.700	-41.800	-36.400	-26.900
euribor3m	41188.0	3.621291	1.734447	0.634	1.344	4.857	4.961	5.045
nr.employed	41188.0	5167.035911	72.251528	4963.600	5099.100	5191.000	5228.100	5228.100

Figure 3.3: Description of numerical features

3. Finally, plot the distributions of the single numerical columns:

```
# plot distributions of numerical features
plt.figure(figsize=(10,18))
for index, col in enumerate(numerical_features):
    plt.subplot(5, 2, index+1)
    sns.distplot(bank_data[col])
plt.savefig("figs/exercise_3_01_distributions.png", \
            format="png", dpi=500)
```

The following is the output of the preceding code:

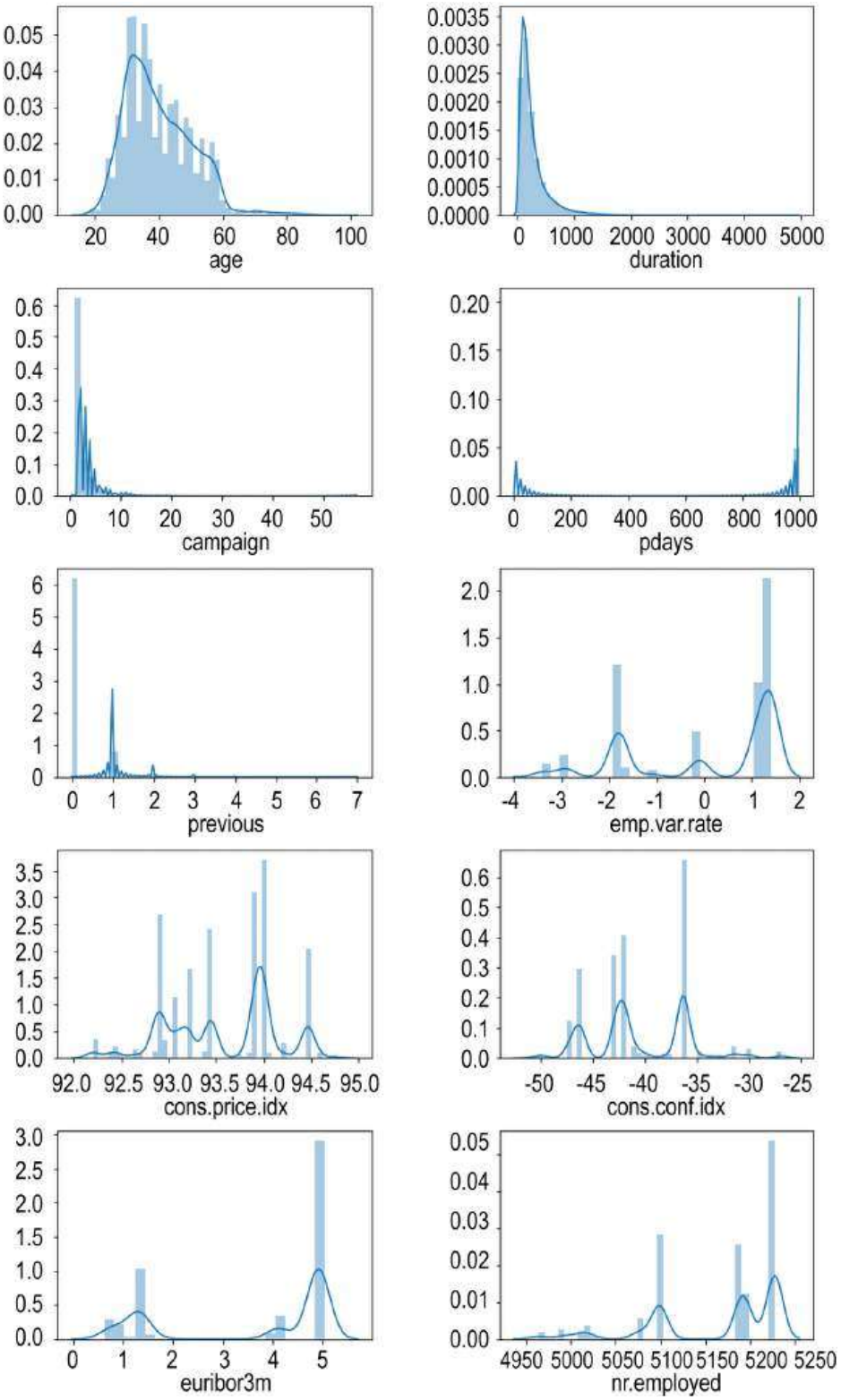

Figure 3.4: Distributions of the numerical features in the banking dataset

> **NOTE**
>
> The above code can sometimes result in an error, depending on the version of statsmodels that is used. If you do encounter an error, you can achieve a similar output by replacing **`sns.distplot(bank_data[col])`** with **`sns.distplot(bank_data[col], kde_kws={'bw': 0.1})`**. This manually passes in a small bandwidth value and avoids the plotting error. There will be small variations between your output and the one above if you use this method.

From the preceding plot, we can see that, besides the **`age`** and **`duration`** columns, the rest of the numerical features do not seem to have a regular distribution of values since their distributions seem scattered and only a few values are present.

> **NOTE**
>
> To access the source code for this specific section, please refer to https://packt.live/30MuxIH.
>
> You can also run this example online at https://packt.live/2Y63IgL. You must execute the entire Notebook in order to get the desired result.

In the following exercise, we will focus on the analysis of the categorical features.

EXERCISE 3.02: ANALYZING DISTRIBUTIONS OF CATEGORICAL FEATURES IN THE BANKING DATASET

In this exercise, you will perform a simple analysis of the categorical features. This will give you an idea of their distributions and the most frequent value in each categorical feature, and, in general, it will serve as a reference for later analysis. You will start by programmatically selecting them, and then create bar plots for the number of entries in each class for each feature. This will give you a general overview of their distributions. Finally, you will explicitly check the number of entries in the **`y`** column, which contains information about whether the marketing campaign was successful or not. Follow these steps to complete this exercise:

1. Select the categorical features in the banking dataset. Note that in **`pandas`**, categorical features are of the **`object`** type and are encoded as strings:

```
# define categorical features
categorical_features = [col for col in bank_data.columns \
                        if pd.api.types\
                        .is_string_dtype(bank_data[col])]
print(categorical_features)
```

   The following is the output of the preceding code:

```
['job', 'marital', 'education', 'default', 'housing', 'loan',
 'contact', 'month', 'day_of_week', 'poutcome', 'y']
```

   You will now plot the distributions of entries among the different features.

2. Use the **`matplotlib.pyplot.subplot()`** function to create multiple plots in the same figure:

```
# plot distributions of numerical features
plt.figure(figsize=(25,35))
for index, col in enumerate(categorical_features):
    plt.subplot(6, 2, index+1)
    ax = sns.countplot(y=col, data=bank_data)
    ax.set_xlabel("count", fontsize=20)
    ax.set_ylabel(col, fontsize=20)
    ax.tick_params(labelsize=20)

plt.savefig("figs/exercise_3_02_counts.png", \
            format="png", dpi=500)
```

The resulting plot is shown in the following output:

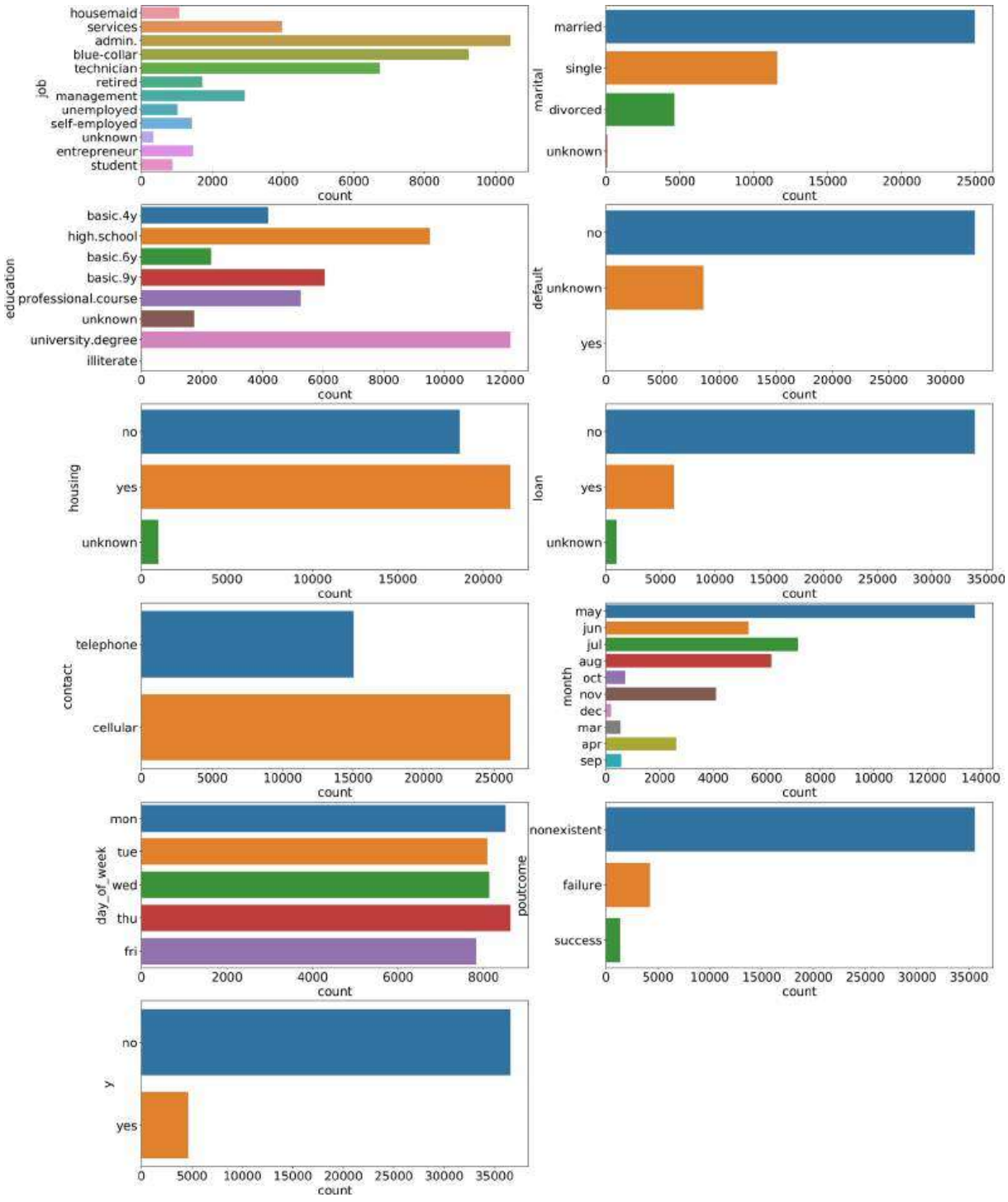

Figure 3.5: Bar plots of the categorical features

3. Finally, explicitly compute the number of entries in the `y` column since it contains information about the outcome of the marketing campaign:

```
# compute number of entries in y column
print("Total number of entries:")
print(bank_data["y"].value_counts(ascending=True))
print()
print("Percentages:")
print(bank_data["y"].value_counts(normalize=True, \
                                  ascending=True)*100)
```

The following is the output of the preceding code:

```
Total number of entries:
yes     4640
no     36548
Name: y, dtype: int64

Percentages:
yes    11.265417
no     88.734583
Name: y, dtype: float64
```

From the preceding output, we can derive that only 11% of the contacted customers decided to accept the offer from the bank.

> **NOTE**
>
> To access the source code for this specific section, please refer to https://packt.live/2B9ieLM.
>
> You can also run this example online at https://packt.live/30HWIbM. You must execute the entire Notebook in order to get the desired result.

In this section, we acquired some basic understanding of the various features (numerical and categorical). This is always important as understanding the basic statistics and distributions of the features serves as a basis for further and more detailed analysis. In the next section, we will analyze the relationships between the feature columns and the outcome of each call (that is, the `y` column).

IMPACT OF NUMERICAL FEATURES ON THE OUTCOME

In this section, we will analyze the relationship between the numerical features (already identified in *Exercise 3.01, Analyzing Distributions of Numerical Features in the Banking Dataset*) and the outcome of a marketing campaign, which is identified in the **`y`** column in the banking dataset.

We will start our analysis by addressing the following question: Is there a statistically significant difference in numerical features for successful and non-successful marketing campaigns? For this reason, we will create violin plots (as shown in the previous chapters) that compare the distribution of the numerical features for the two types of outcomes ("yes" for a successful marketing campaign, "no" for an unsuccessful one):

```
"""
create violin plots for successful and non-successful marketing campaigns
"""
plt.figure(figsize=(10,18))
for index, col in enumerate(numerical_features):
    plt.subplot(5, 2, index+1)
    sns.violinplot(x=col, y="y", data=bank_data, \
                   order=["yes", "no"])
plt.savefig("figs/violin_plots_numerical_features.png", \
            format="png", dpi=500)
```

The following is the output of the preceding code:

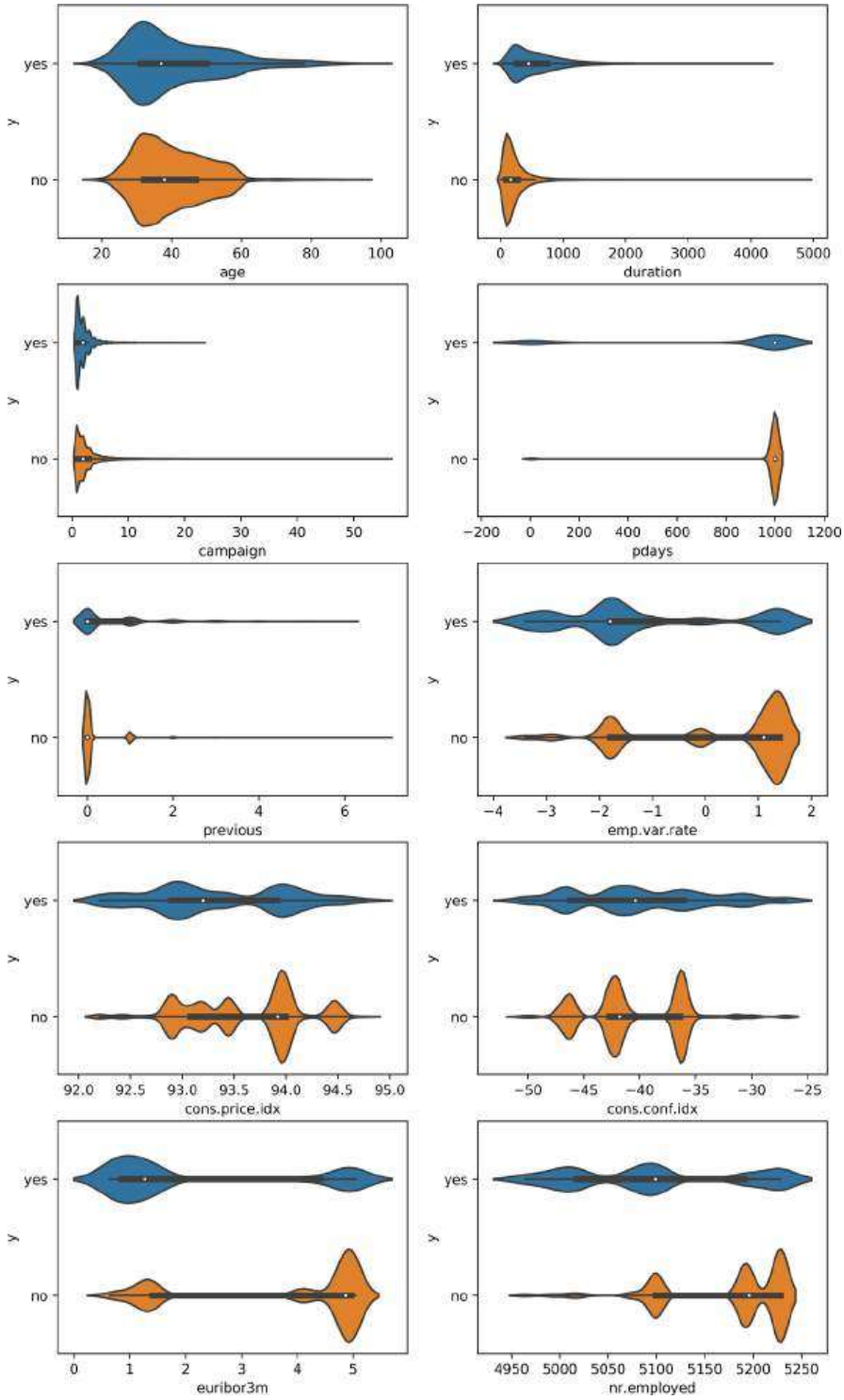

Figure 3.6: Violin plots of numerical features against the outcome of the marketing campaign

The preceding figure shows that the distributions of most of the numerical features vary between successful marketing campaigns (those whose **y** column value is equal to "yes") and unsuccessful ones (those whose **y** column value is equal to "no"). We can assess this difference from a statistical perspective by running a hypothesis test on each of the numerical features, in which we will test whether the average value of the respective numerical feature is different for "yes" entries for the **y** column against "no" entries for each numerical feature.

First, let's define a function that returns the average values of the column in the "yes" and "no" groups for the provided column, as well as the **test statistic** and **p-value** from the equality of means test:

```
from scipy.stats import ttest_ind
"""
define function for computing mean of column for yes and no cases,
as well as the test statistics and pvalue for equality of means test
"""
def test_means(data, col):
    yes_mask = data["y"] == "yes"
    values_yes = data[col][yes_mask]
    values_no = data[col][~yes_mask]
    mean_yes = values_yes.mean()
    mean_no = values_no.mean()

    ttest_res = ttest_ind(values_yes, values_no)

    return [col, mean_yes, mean_no, \
            round(ttest_res[0], 4), round(ttest_res[1],4)]
```

Then, we define a **pandas** DataFrame that will contain the values that were returned by the **test_means** function:

```
# define pandas dataframe, in which values should be filled
test_df = pd.DataFrame(columns=["column", "mean yes", \
                                "mean no", "ttest stat", \
                                "ttest pval"])
"""
for each column in the numerical_features, compute means and test
statistics and fill the values in the dataframe
"""
for index, col in enumerate(numerical_features):
    test_df.loc[index] = test_means(bank_data, col)

test_df
```

The results of the preceding code snippet can be seen in the following output:

	column	mean yes	mean no	ttest stat	ttest pval
0	age	40.913147	39.911185	6.1721	0.0
1	duration	553.191164	220.844807	89.9672	0.0
2	campaign	2.051724	2.633085	-13.4965	0.0
3	pdays	792.035560	984.113878	-69.7221	0.0
4	previous	0.492672	0.132374	48.0027	0.0
5	emp.var.rate	-1.233448	0.248875	-63.4337	0.0
6	cons.price.idx	93.354386	93.603757	-27.9032	0.0
7	cons.conf.idx	-39.789784	-40.593097	11.1539	0.0
8	euribor3m	2.123135	3.811491	-65.6466	0.0
9	nr.employed	5095.115991	5176.166600	-76.9845	0.0

Figure 3.7: Difference in mean and results of the equality of means test for numerical columns

As we can see from the previous plot, there is a statistically significant difference in the mean values for each of the numerical columns (the results from the p-value in the **`ttest pval`** column). This means that for each of the numerical features, the average value for successful marketing campaigns is significantly different than the average value for unsuccessful marketing campaigns.

We can not only analyze the difference in means but also the distribution difference for each numerical column. We will do this in the following exercise.

EXERCISE 3.03: HYPOTHESIS TEST OF THE DIFFERENCE OF DISTRIBUTIONS IN NUMERICAL FEATURES

In this exercise, you will programmatically test the difference between the distributions of successful and unsuccessful marketing calls for each numerical feature. This is a good way to structure algorithmic-based reasoning and perform hypothesis testing on a variety of features. More precisely, you will perform a Kolmogorov-Smirnov test for the equality of distributions for each of the numerical features for successful versus unsuccessful marketing campaigns. Follow these steps to complete this exercise:

1. First, define a function that, for a provided column, computes the Kolmogorov-Smirnov test and returns the values in an array:

```
from scipy.stats import ks_2samp

"""
define function which performs Kolmogorov-Smirnov test,
for provided column
"""
def test_ks(data, col):
    yes_mask = data["y"] == "yes"
    values_yes = data[col][yes_mask]
    values_no = data[col][~yes_mask]

    kstest_res = ks_2samp(values_yes, values_no)
    return [col, round(kstest_res[0], 4), \
            round(kstest_res[1],4)]
```

2. Next, define a **`pandas`** DataFrame that contains the results from the function defined in *Step 1* by applying it to each numerical feature:

```
# define pandas dataframe, in which values should be filled
test_df = pd.DataFrame(columns=["column", "ks stat", "ks pval"])

"""
for each column in the numerical_features,
compute test statistics and fill the values in the dataframe
"""
for index, col in enumerate(numerical_features):
    test_df.loc[index] = test_ks(bank_data, col)

test_df
```

The following is the output of the preceding code:

	column	ks stat	ks pval
0	age	0.0861	0.0
1	duration	0.4641	0.0
2	campaign	0.0808	0.0
3	pdays	0.1934	0.0
4	previous	0.2102	0.0
5	emp.var.rate	0.4324	0.0
6	cons.price.idx	0.2281	0.0
7	cons.conf.idx	0.1998	0.0
8	euribor3m	0.4326	0.0
9	nr.employed	0.4324	0.0

Figure 3.8: Results from the Kolmogorov-Smirnov test for equality of distributions

From the preceding figure, we can also observe that the distributions of the various numerical features present a significant difference between successful and unsuccessful marketing campaigns.

> **NOTE**
>
> To access the source code for this specific section, please refer to https://packt.live/2CdYyqH.
>
> You can also run this example online at https://packt.live/3e7FU1v. You must execute the entire Notebook in order to get the desired result.

In order to further deepen our analysis, also create pairplots between the numerical features and divide the groups by the outcome of the marketing campaign. First, let's divide the numerical features into two groups: campaign-related features and financial features. In the first group, we will include the **`age`**, **`duration`**, **`campaign`**, and **`previous`** columns, while in the latter one, we will include the **`emp.var.rate`**, **`cons.price.idx`**, **`cons.conf.idx`**, and **`euribor3m`** columns:

```
# create arrays containing campaign and financial columns
campaign_columns = ["age", "duration", "campaign", "previous"]
financial_columns = ["emp.var.rate", "cons.price.idx", \
                     "cons.conf.idx", "euribor3m"]
```

The following code snippet plots the campaign columns:

```
# create pairplot between campaign columns
plot_data = bank_data[campaign_columns + ["y"]]
plt.figure(figsize=(10,10))
sns.pairplot(plot_data, hue="y", palette="bright")
plt.savefig("figs/pairplot_campaign.png", \
            format="png", dpi=300)
```

The following is the output of the preceding code:

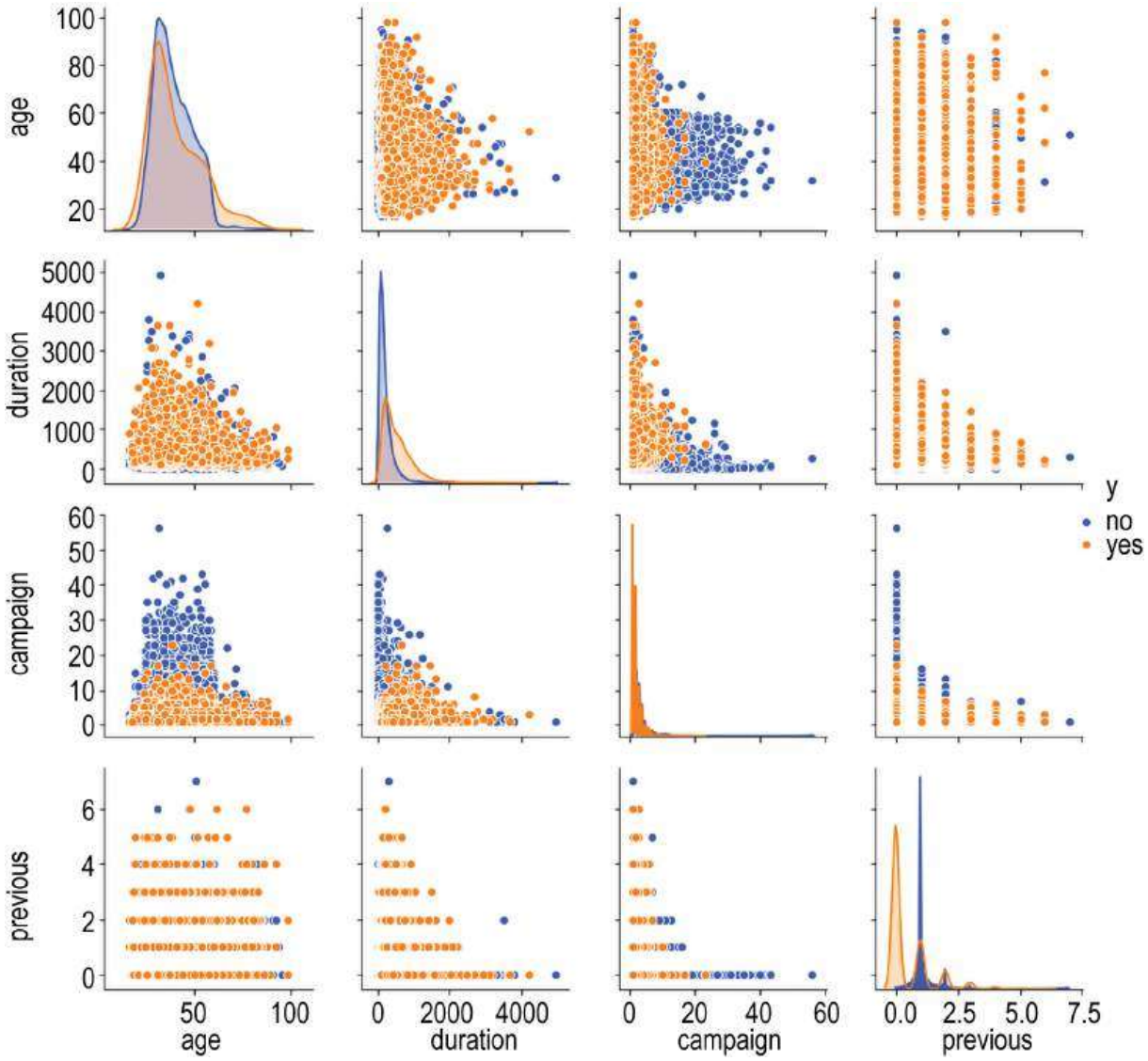

Figure 3.9: Pairplot of the campaign features, grouped by marketing outcome

> **NOTE**
>
> As in the earlier exercise, if you get an error due to a known issue with the packages used, you can get a similar output by using **`sns.pairplot(plot_data, hue="y", palette="bright", diag_kws={'bw': 0.1})`**. This will give a slightly different output due to the manual bandwidth setting, but the plot shapes should be a similar shape to those shown above.

From the preceding figure, we can immediately see that, in the **`previous`** column, most of the successful marketing campaigns were with newly contacted customers, while a substantial peak is present for customers who were contacted the second time, but without success.

Now, consider the financial columns:

```
# create pairplot between financial features
plot_data = bank_data[financial_columns + ["y"]]
plt.figure(figsize=(10,10))
sns.pairplot(plot_data, hue="y", palette="bright")
plt.savefig("figs/pairplot_financial.png", \
            format="png", dpi=300)
```

The following is the output of the preceding code:

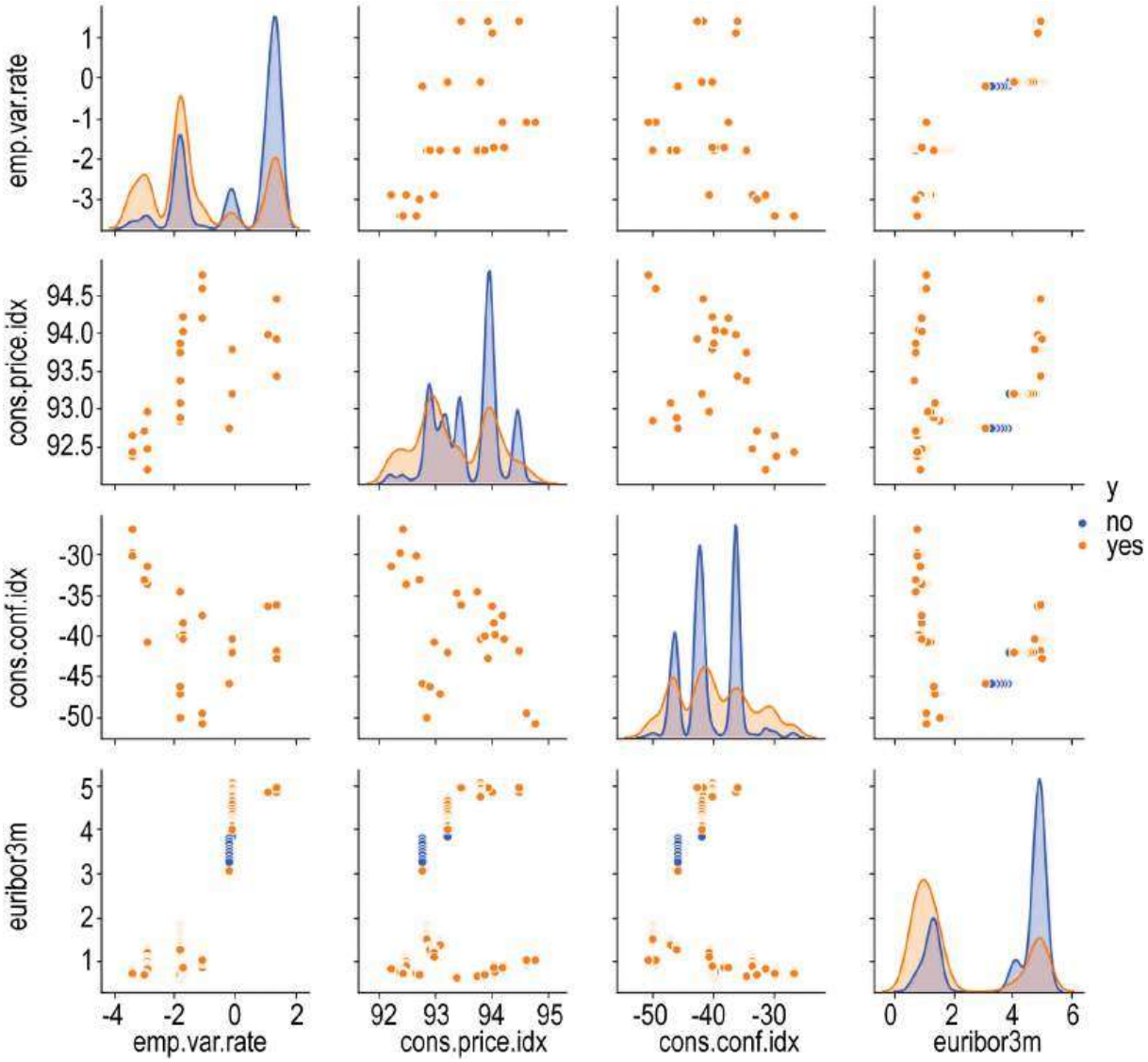

Figure 3.10: Pairplot of the financial features, grouped by marketing outcome

Interestingly, for lower values for the 3-month interest rates (the **`euribor3m`** column), the number of successful marketing calls is larger than the number of unsuccessful ones. The inverse situation happens when interest rates are higher. A possible explanation for this phenomenon is customer optimism when interest rates are lower. You will investigate the market indicators and their impact on the outcome of the marketing calls later in this section.

Analyze the correlations between the different numerical features. This is important so that we have a clear picture of how the different features behave with respect to each other. We will do this by distinguishing between the two cases, that is, successful and unsuccessful customer calls:

```
# create mask for successful calls
successful_calls = bank_data.y == "yes"

# plot correlation matrix for successful calls
plot_data = bank_data[campaign_columns + financial_columns]\
                     [successful_calls]
successful_corr = plot_data.corr()
successful_corr.style.background_gradient(cmap='coolwarm')\
.set_precision(2)
```

The following is the output of the preceding code:

	age	duration	campaign	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
age	1	-0.059	-0.0079	0.075	-0.082	-0.025	0.14	-0.085
duration	-0.059	1	0.16	-0.23	0.5	0.24	-0.14	0.5
campaign	-0.0079	0.16	1	-0.1	0.22	0.12	-0.043	0.21
previous	0.075	-0.23	-0.1	1	-0.28	0.091	0.13	-0.39
emp.var.rate	-0.082	0.5	0.22	-0.28	1	0.66	-0.27	0.93
cons.price.idx	-0.025	0.24	0.12	0.091	0.66	1	-0.33	0.41
cons.conf.idx	0.14	-0.14	-0.043	0.13	-0.27	-0.33	1	-0.12
euribor3m	-0.085	0.5	0.21	-0.39	0.93	0.41	-0.12	1

Figure 3.11: Correlation matrix of numerical features for successful customer calls

In the preceding figure, we can see the correlation matrix of the numerical features, filtered only for successful customer calls. Do the same for unsuccessful calls:

```
# plot correlation matrix for unsuccessful calls
plot_data = bank_data[campaign_columns + financial_columns]\
                     [~successful_calls]
unsuccessful_corr = plot_data.corr()
unsuccessful_corr.style.background_gradient(cmap='coolwarm')\
.set_precision(2)
```

The following is the output of the preceding code:

	age	duration	campaign	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
age	1	0.0008	0.0084	-0.003	0.027	0.011	0.12	0.041
duration	0.0008	1	-0.083	-0.0043	0.0035	0.018	0.0042	0.0065
campaign	0.0084	-0.083	1	-0.068	0.13	0.12	-0.0071	0.12
previous	-0.003	-0.0043	-0.068	1	-0.42	-0.27	-0.14	-0.44
emp.var.rate	0.027	0.0035	0.13	-0.42	1	0.8	0.32	0.98
cons.price.idx	0.011	0.018	0.12	-0.27	0.8	1	0.15	0.73
cons.conf.idx	0.12	0.0042	-0.0071	-0.14	0.32	0.15	1	0.39
euribor3m	0.041	0.0065	0.12	-0.44	0.98	0.73	0.39	1

Figure 3.12: Correlation matrix of numerical features for unsuccessful customer calls

As we can observe from the preceding two figures, the correlation between **`euribor3m`** and **`emp.var.rate`** is very high (approximately 0.93 for successful and 0.98 for unsuccessful calls). That is quite an interesting phenomenon as the first one relates to the average interest rate at which European banks lend money to other banks with a maturity of 3 months, while the second one relates to the employment variation, that is, the rate at which people are hired or fired in an economy.

A positive correlation between those values means that in an expanding economy, in which more people are hired than fired (hence, the positive values in **`emp.var.rate`**), interest rates tend to be higher. Another column that is also highly correlated with the previous two is the **Consumer Price Index** (**CPI**) column: **`cons.price.idx`**.

By definition (https://www.investopedia.com/terms/c/consumerpriceindex.asp), the CPI is a measure that observes the change of prices of a basket of consumers' goods and services (food, transportation, households, and medical care). The basket itself is a collection of goods and services from eight major areas: medical care, food and beverages, housing, apparel, transportation, education, recreation, and other goods and services. The formula for the CPI is as follows:

$$CPI = \frac{Cost\ of\ \ Market\ Basket\ in\ Given\ Year}{Cost\ of\ \ Market\ Basket\ in\ Base\ Year} \times 100$$

Figure 3.13: Formula for the CPI

In the preceding equation, the cost of the market basket is compared between two different years. Changes in the CPI are related to changes in the cost of living and are therefore a clear indicator of inflation or deflation.

A high positive correlation of the **`cons.price.idx`** column with the **`emp.var.rate`** and **`euribor3m`** columns is a clear indicator of an expanding economy, in which growing employment means more people with money—hence more spending and higher inflation. In such situations, central banks tend to increase interest rates (in order to keep inflation under control), which is totally in line with our observation of increasing the **`euribor3m`** column.

Finally, returning to our marketing campaign problem, consider the difference between the correlation matrices for successful and unsuccessful calls:

```
"""
plot difference of successful - unsuccessful correlation matrices
"""
diff_corr = successful_corr - unsuccessful_corr
diff_corr.style.background_gradient(cmap='coolwarm')\
.set_precision(2)
```

The following is the output of the preceding code:

	age	duration	campaign	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
age	0	-0.06	-0.016	0.078	-0.11	-0.036	0.016	-0.13
duration	-0.06	0	0.24	-0.23	0.5	0.22	-0.15	0.49
campaign	-0.016	0.24	0	-0.036	0.089	-0.0068	-0.036	0.097
previous	0.078	-0.23	-0.036	0	0.14	0.36	0.27	0.05
emp.var.rate	-0.11	0.5	0.089	0.14	0	-0.14	-0.59	-0.05
cons.price.idx	-0.036	0.22	-0.0068	0.36	-0.14	0	-0.48	-0.32
cons.conf.idx	0.016	-0.15	-0.036	0.27	-0.59	-0.48	0	-0.51
euribor3m	-0.13	0.49	0.097	0.05	-0.05	-0.32	-0.51	0

Figure 3.14: Difference between correlation matrices for successful and unsuccessful calls

In the preceding figure, you can see the difference between the correlation matrices, one for successful calls and one for unsuccessful calls. The values indicated in the matrix can provide us with information about strong differences in correlations between successful and unsuccessful calls. You can immediately see that there is a significant difference in the correlations between the **`duration`** and **`emp.var.rate`** columns for successful (correlation: **`0.5`**) and unsuccessful (correlation: **`0.0035`**) calls.

This might look like a significant indicator at first, but note that the **`duration`** column represents the duration of the call. Therefore, we can expect successful calls to have a longer duration (as information about the product has to be presented and the customer's data has to be gathered). On the other hand, the **`emp.var.rate`** column represents the employment variation rate, a macroeconomic factor that is definitely not influenced by the length of certain phone calls. It is always important to remember that correlation does not imply causation. Another interesting relationship is the one between the **`cons.conf.idx`** column and the **`emp.var.rate`**, **`cons.price.idx`**, and **`euribor3m`** columns.

The **`cons.conf.idx`** column represents the Consumer Confidence Index, an economic indicator that defines the degree of optimism in an economy. The correlations between the **`cons.conf.idx`** and **`emp.var.rate`** columns and the **`cons.price.idx`** and **`euribor3m`** columns for successful and unsuccessful calls are given in the following table:

Columns	Successful Calls	Unsuccessful Calls
cons.conf.idx – emp.var.rate	-0.27	0.32
cons.conf.idx – cons.price.idx	-0.33	0.15
cons.conf.idx – euribor3m	-0.12	0.39

Figure 3.15: Correlations between the columns

This is quite an interesting phenomenon as the **`emp.var.rate`**, **`cons.price.idx`**, and **`euribor3m`** columns are strongly correlated. We already mentioned that during periods of expanding economy, the employment rate increases, leading to higher consumer price index and higher interest rates (the situation is the inverse in contracting economy and periods of depression).

From the preceding table, an interesting fact arises: the consumer confidence index is negatively correlated with the three mentioned columns for successful customer calls, and positively correlated for unsuccessful ones. This means that when the overall economic sentiment is pessimistic, people are willing to accept the new banking products and vice versa.

MODELING THE RELATIONSHIP VIA LOGISTIC REGRESSION

We will dedicate the rest of this chapter to one of the fundamental techniques in data analysis and machine learning: linear and logistic regression.

Suppose that we are estimating the relationship between m different features.

In both linear and logistic regression, a set of features, $\mathbf{X}_1, \ldots, \mathbf{X}_m$, and a target variable, $\mathbf{Y}$, are provided to model the target variable, $\mathbf{Y}$, as a function of the features, $\mathbf{X}_1, \ldots, \mathbf{X}_m$, as in the following equation:

$$Y \approx f(X_1, \ldots, X_m)$$

Figure 3.16: General form of linear/logistic regression

LINEAR REGRESSION

In linear regression, the target variable, Y, is a continuous variable, meaning that it assumes all possible values in a bounded or unbounded interval, (a,b) $\subseteq$ R, where R is the set of real numbers. In this way, the preceding equation assumes the following concrete form:

$$Y \approx \alpha_0 + \sum_{j=1}^{m} \alpha_j X_j$$

Figure 3.17: Linear regression equation

Let's denote the right-hand side of the preceding equation with $\hat{\mathbf{Y}}$, as follows:

$$\hat{Y} = \alpha_0 + \sum_{j=1}^{m} \alpha_i X_j$$

Figure 3.18: Linear regression equation

Then, if we have n samples in our data (where for each i ϵ {1,..., n}, we denote the entries for the m features with $x_{i,1}$,....$x_{i,m}$ and the target variable with y_i), we can rewrite the previous equation in a more specific form, as follows:

$$y_i \approx \hat{y}_i = \alpha_0 + \sum_{j=1}^{m} \alpha_j X_{i,j}$$

Figure 3.19: Linear regression equation in a specific form

Note that in *Figure 3.17* and *Figure 3.19*, we assume that the dependency of Y from the feature vectors X_1,...,X_m is either linear or can be approximated with a linear equation. With this assumption, estimating the relationship between Y and the features X_1,...,X_m means finding the parameters $\alpha_{o, ..., } \alpha_m$ so that the "distance" between Y and $\hat{Y}$ is minimized. One of the most common methods that's used to minimize the distance between the two vectors Y and $\hat{Y}$ is the *least squares* method, which aims to find the coefficients $\alpha = (\alpha_0, ..., \alpha_m)$ so that the residual sum of squares is minimized:

$$RSS(\alpha) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \alpha_0 - \sum_{j=1}^{m} \alpha_j X_{i,j} \right)^2$$

Figure 3.20: Residual sum of squares for linear regression

In order to solve the preceding equation, we need to introduce the matrix notations of the target variable, Y, the feature matrix, X, and the parameter vector, **α**, which are defined as follows:

$$Y = \begin{bmatrix} y_1 \\ . \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 x_{11} & \dots & x_{1m} \\ \dots & & \\ 1 x_{n1} & \dots & x_{nm} \end{bmatrix} \quad \alpha = \begin{bmatrix} \alpha_1 \\ . \\ \alpha_m \end{bmatrix}$$

Figure 3.21: Definitions of the matrix notation of Y, X, and α

With these notations, *Figure 3.20* becomes the following, where X^T denotes the transpose of the matrix, X:

$$RSS(\alpha) = \left(Y - X_\alpha\right)^T \left(Y - X_\alpha\right)$$

Figure 3.22: Expression for RSS(α)

In order to compute the values of the vector, **α**, that minimize the equation in *Figure 3.19*, we need to impose the conditions of local minima for the function RSS(.) with respect to **α** —that is, that the gradient of RSS is equal to zero and its Hessian matrix (meaning the second derivative of the function RSS with respect to the vector, **α**) is positive definite. In other words, the following conditions have to be imposed on **α**, in order to minimize the equation in *Figure 3.20*:

$$\frac{\partial RSS}{\partial \alpha}(\alpha) = -2X^T(y - X\alpha) = 0$$

Figure 3.23: Condition of the gradient of RSS to be equal to zero

$$\frac{\partial^2 RSS}{\partial \alpha^2}(\alpha) = 2X^T X \text{ positive definite}$$

Figure 3.24: Condition of the hessian matrix of RSS to be positive definite

Note that from the second conditions (*Figure 3.24*), it follows that the matrix X^TX is invertible with the inverse matrix $(X^TX)^{-1}$. We can rewrite the equation in *Figure 3.23* as follows:

$$\hat{\alpha} = (X^T X)^{-1} X^T y$$

Figure 3.25: Analytical solution of equation in Figure 3.20

This is the value of **α**, which minimizes the function in the right-hand side of the equation in *Figure 3.20*.

From a data analysis perspective, the computation of the preceding matrices is performed automatically in various Python packages, leaving the analyst the more interesting task of analyzing the data and drawing important business conclusions, rather than implementing the matrix multiplications personally. Let's see how we can apply linear regression to our banking data in order to create a linear model that's able to identify the relationship between the various numerical features.

In this section, we will be applying linear regression in order to predict the consumer confidence index (the **`cons.conf.idx`** column) based on the employment variation rate, the CPI, and the **`euribor`** 3-month interest rate (the **`emp.var.rate`**, **`cons.price.idx`**, and **`euribor3m`** columns, respectively). In Python, we can easily fit a linear regression model by using the **`OLS()`** function from the **`statsmodels.api`** package:

```
import statsmodels.api as sm

# create feature matrix and target variable
X = bank_data[["emp.var.rate", "cons.price.idx", "euribor3m"]]
# add constant value for the intercept term
X = sm.add_constant(X)
y = bank_data["cons.conf.idx"]

# define and fit model
lineare_regression_model = sm.OLS(y, X)
result = lineare_regression_model.fit()
print(result.summary())
```

The summary of the result is shown in the following figure:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          cons.conf.idx   R-squared:                       0.177
Model:                            OLS   Adj. R-squared:                  0.177
Method:                 Least Squares   F-statistic:                     2960.
Date:                Mon, 10 Feb 2020   Prob (F-statistic):               0.00
Time:                        23:28:51   Log-Likelihood:            -1.1753e+05
No. Observations:               41188   AIC:                         2.351e+05
Df Residuals:                   41184   BIC:                         2.351e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const            -82.4025      5.999    -13.736      0.000     -94.161     -70.644
emp.var.rate      -4.1814      0.072    -57.960      0.000      -4.323      -4.040
cons.price.idx     0.2828      0.063      4.478      0.000       0.159       0.407
euribor3m          4.3582      0.057     76.618      0.000       4.247       4.470
==============================================================================
Omnibus:                     3246.559   Durbin-Watson:                   0.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4034.493
Skew:                           0.761   Prob(JB):                         0.00
Kurtosis:                       2.811   Cond. No.                     2.72e+04
==============================================================================
```

Figure 3.26: Results from the OLS model

Several statistics are provided in the result from the **`summary()`** function. Some of the most important ones are the **`R-squared`** and **`Adj. R-squared`** metrics and the **`coef`** and **`P>| t |`** columns. Let's investigate those values in more detail.

The **`R-squared`** metric, also known as **coefficient of determination**, is the proportion of variance in the dependent variable (**`cons.conf.idx`**, in our case) that is predicted by our model. From a mathematical perspective, it can be computed by the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

Figure 3.27: Definition of R^2

Here, $\hat{y}_i$ is the predicted value by the model and can be expressed as follows:

$$\hat{y}_i = \alpha_0 + \sum_{j=1}^{m} \alpha_j X_{i,j}$$

Figure 3.28: Definition of $\hat{y}_i$

That $\bar{y}$ is the average value of y and can be expressed as follows:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Figure 3.29: Definition of $\bar{y}$

In general, the R-squared metric is limited in the interval [0, 1], where R-squared values close to 0 mean predicting the average value of the target variable (thus, almost no predictive power), while values close to 1 indicate a very accurate prediction model. One interesting property of the R-squared metric is that it tends to increase by increasing the complexity of the model (that is, adding more features). This doesn't always mean that our model is becoming more accurate as sometimes, those features are completely irrelevant. For this reason, we often tend to consider the **Adjuster R-squared** metric, which not only takes into account the accuracy of the model but also its complexity:

$$\overline{R^2} = 1 - (1 - R^2) \frac{n-1}{n-m-1}$$

Figure 3.30: Definition of $\overline{R^2}$

Now, let's consider the **coef** column in *Figure 3.25*. It provides the coefficients of the linear regression formula in *Figure 3.17*. More precisely, our model assumes the following form:

$$cons.conf.idx = -82.4025 - 4.1814 \cdot emp.var.rate + 0.2828 \cdot cons.price.idx + 4.3582 \cdot euribor3m$$

Figure 3.31: Linear regression model of cons.conf.idx as a function of emp.var.rate, cons. price.idx, and euribor3m

The single values of the coefficients are quite important. They tell us how much **cons.conf.idx** will increase if we increase one of the variables by 1 while keeping the other constant. For instance, the **cons.conf.idx** value will increase by 4.3582, if the value of the **euribor3m** is increased by 1, while it will decrease by 4.1814 at any increase of **emp.var.rate**. This means that **cons.conf.idx** is positively correlated with the **cons.price.idx** and **euribor3m** columns, and negatively correlated with **emp.var.rate**.

Finally, the **P>| t |** column in *Figure 3.26* returns the p-value of a hypothesis test, in which the null hypothesis is that the relative coefficient is equal to zero. In *Figure 3.26*, we can see that all those p-values are 0, meaning that each of the coefficients is statistically significant in our equation. Sometimes, these p-values become large, which means that the feature associated with that value is not relevant for our linear regression model and can be removed from the equation, without deteriorating the accuracy of the model.

LOGISTIC REGRESSION

Logistic regression is very similar to the linear regression technique we introduced in the previous section, with the only difference that the target variable, **Y**, assumes only values in a discrete set; say, for simplicity {0, 1}. If we were to approach such a problem as a logistic regression problem, the output of the right-hand side of the equation in *Figure 3.17* could easily go way beyond the values 0 and 1. Furthermore, even by limiting the output, it will still be able to assume all the values in the interval [0, 1]. For this reason, the idea behind logistic regression is to model the *probability* of the target variable Y, to assume one of the values (say 1). In this case, all the values between 0 and 1 will be reasonable.

With **p**, let's denote the probability of the target variable, **Y**, being equal to 1 when it's given a specific feature **x**:

$$p = Pr(Y = y | X = x)$$

Figure 3.32: Definition of p

Let's also define the **logit** function:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Figure 3.33: Definition of the logit function

Note that the **logit** function maps the interval (0, 1) into $(-\infty, +\infty)$, as shown in *Figure 3.32*, as generated by the following code snippet:

```
# plot logit function
x = np.arange(0, 1, 0.01)
logit = np.log(x/(1-x))

plt.figure(figsize=(6,6))
plt.plot(x,logit)
plt.xlabel("p")
plt.ylabel("$\log(\\frac{p}{1-p})$")
plt.grid()
plt.savefig("figs/logit_function.png", \
            format="png", dpi=300)
```

The output will be as follows:

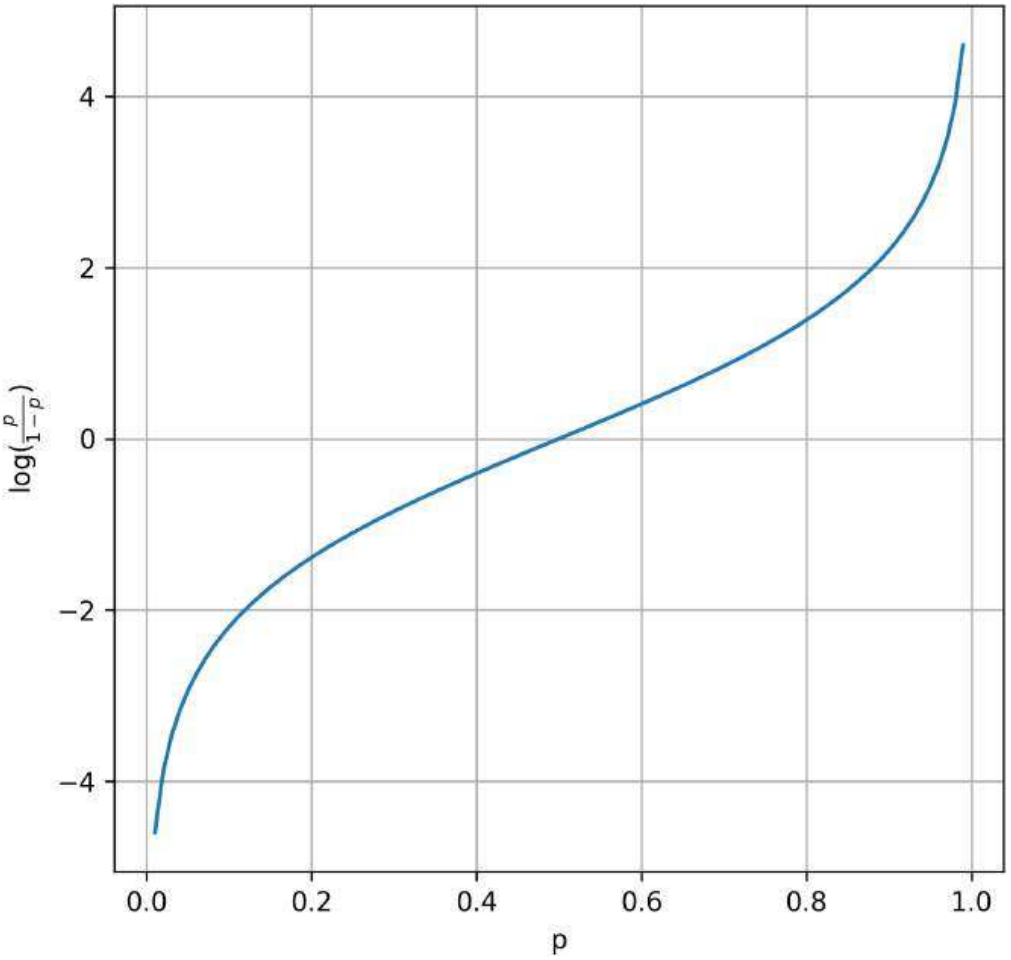

Figure 3.34: The logit function

With the help of the **`logit`** function, we can transform the output from the linear model in the equation in *Figure 3.17* into a probability:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \alpha_0 + \sum_{j=1}^{m} \alpha_j X_j$$

Figure 3.35: Transforming output into probability

By solving the last equation with respect to p, we can obtain the estimation of the probability of the target variable, Y, being equal to 1, given the feature vector **`X = x`**, as follows:

$$Pr(Y=1|X=x) = \frac{exp(\alpha_0 + \sum_{j=1}^{m} \alpha_j X_j)}{1 + exp(\alpha_0 + \sum_{j=1}^{m} \alpha_j X_j)}$$

Figure 3.36: Probability estimation via logistic regression (Y = 1)

$$Pr(Y=0|X=x) = \frac{1}{1 + exp(\alpha_0 + \sum_{j=1}^{m} \alpha_j X_j)}$$

Figure 3.37: Probability estimation via logistic regression (Y = 0)

As is the case of linear regression, Python's **`statsmodels.api`** package already contains a function that implements the logistic regression model. Let's look at how we can use this in order to create a logistic regression model that has the **`age`**, **`duration`**, **`campaign`**, and **`previous`** columns as feature vectors and the **`y`** column as the target variable:

```
# create feature matrix and target variable
X = bank_data[["age", "duration", "campaign", "previous"]]
# add constant value for the intercept term
X = sm.add_constant(X)
# target has to be numeric
y = np.where(bank_data["y"] == "yes", 1, 0)

# define and fit model
logistic_regression_model = sm.Logit(y, X)
result = logistic_regression_model.fit()
print(result.summary())
```

The result of the **`summary()`** function can be seen in the following figure:

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                41188
Model:                          Logit   Df Residuals:                    41183
Method:                           MLE   Df Model:                            4
Date:                Tue, 11 Feb 2020   Pseudo R-squ.:                  0.2331
Time:                        17:19:35   Log-Likelihood:                -11119.
converged:                       True   LL-Null:                       -14499.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.7793      0.076    -49.435      0.000      -3.929      -3.629
age            0.0091      0.002      5.665      0.000       0.006       0.012
duration       0.0039   6.16e-05     62.973      0.000       0.004       0.004
campaign      -0.1163      0.011    -10.706      0.000      -0.138      -0.095
previous       1.0579      0.027     39.232      0.000       1.005       1.111
==============================================================================
```

Figure 3.38: Results from the logistic regression model

As in the case of linear regression, the model returns the parameters from the logistic regression equation. Hence, we can easily construct the estimation of the probabilities of the **`y`** variable being equal to "yes" or "no" as follows:

$$Pr(\text{y = yes}) = \frac{exp(-3.7793 + 0.0091.age + 0.0039.duration - 0.1163.campaign + 1.0579.previous)}{1 + exp(-3.7793 + 0.0091.age + 0.0039.duration - 0.1163.campaign + 1.0579.previous)}$$

Figure 3.39: Probability of the y column being equal to "yes," according to the logistic regression model

$$Pr(\text{y=yes}) = \frac{1}{1 + exp(-3.7793 + 0.0091.age + 0.0039.duration - 0.1163.campaign + 1.0579.previous)}$$

Figure 3.40: Probability of the y column being equal to "no," according to the logistic regression model

Furthermore, from the **`P>|z|`** column in *Figure 3.38*, we can see that all the parameters are statistically significant for our probability estimation.

One important aspect of both the linear and logistic regression models is that both the feature and target vectors must be numeric variables. We already saw how to transform the target variable from categorical into numerical values using the **`numpy.where()`** function:

```
# target has to be numeric
y = np.where(bank_data["y"] == "yes", 1, 0)
```

We can also use the **`apply()`** function on a **`pandas.Series`** object if we need more sophisticated mappings. None of these methods scale well when we have lots of features with categorical values. In order to solve this problem, we can use the **`pandas.get_dummies()`** function, which automatically creates dummy variables for all non-numeric columns. With this technique (also known as **one-hot encoding**), we convert each categorical column into a set of new columns, each one indicating that the original one contains a specific value. Let's provide a simple example. Consider the **`education`** column in the bank marketing campaign. Originally, it contains 8 distinct classes, as shown in the following code:

```
print(bank_data["education"].unique())
```

The output will be as follows:

```
['basic.4y' 'high.school' 'basic.6y' 'basic.9y'
 'professional.course' 'unknown' 'university.degree' 'illiterate']
```

By applying the **`pandas.get_dummies()`** function to the **`education`** column, we get 8 new columns with 0 or 1 as the only possible values, depending on the value in the original **`education`** column. Here is the output of the **`pandas.get_dummies()`** function when applied to the **`education`** column (we also attach the original column for better understanding):

```
hot_encoded = pd.get_dummies(bank_data["education"])
hot_encoded["education"] = bank_data["education"]
hot_encoded.head(10)
```

The following is the output of the preceding code:

	basic.4y	basic.6y	basic.9y	high.school	illiterate	professional.course	university.degree	unknown	education
0	1	0	0	0	0	0	0	0	basic.4y
1	0	0	0	1	0	0	0	0	high.school
2	0	0	0	1	0	0	0	0	high.school
3	0	1	0	0	0	0	0	0	basic.6y
4	0	0	0	1	0	0	0	0	high.school
5	0	0	1	0	0	0	0	0	basic.9y
6	0	0	0	0	0	1	0	0	professional.course
7	0	0	0	0	0	0	0	1	unknown
8	0	0	0	0	0	1	0	0	professional.course
9	0	0	0	1	0	0	0	0	high.school

Figure 3.41: Result of the get_dummies() function being applied to the education column

As shown in the preceding figure, eight new columns have been created, all of which have values equal to 0, except the column corresponding to the original value in the **`education`** column, which has a value of 1.

In the following exercise, we will apply the techniques we've presented so far to our bank marketing campaign data.

EXERCISE 3.04: LOGISTIC REGRESSION ON THE FULL MARKETING CAMPAIGN DATA

In this exercise, you will create a logistic regression model based on the full marketing campaign data. By doing this, you will be able to create a mathematical model that explains the relationship between the different features in the data and the final outcome of the marketing campaign. This could serve as a basis for marketing campaign optimization (for instance, contacting only customers with certain characteristics). In this way, not only could the outcome be improved, but also marketing campaign costs could be reduced.

Although creating and fitting the model is straightforward, you will need to transform all the features into numerical ones by using the **`pandas.get_dummies()`** function. Follow these steps to complete this exercise:

1. First, preprocess the feature matrix so that it contains only numerical values. This is not a problem for columns such as **`age`** or **`duration`** as they already contain numerical values, but columns such as **`job`** or **`month`** should be preprocessed. For this reason, apply the **`pandas.get_dummies()`** function:

```
    """
    transform all features into numerical ones, by using
    the get_dummies() function
    """
    X = bank_data.drop("y", axis=1)
    X = pd.get_dummies(X)
    X = sm.add_constant(X)
    print(X.columns)
```

The output will be as follows:

```
Index(['const', 'age', 'duration', 'campaign', 'pdays', 'previous',
       'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',
       'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',
       'job_services', 'job_student', 'job_technician', 'job_unemployed',
       'job_unknown', 'marital_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_basic.4y', 'education_basic.6y',
       'education_basic.9y', 'education_high.school', 'education_illiterate',
       'education_professional.course', 'education_university.degree',
       'education_unknown', 'default_no', 'default_unknown', 'default_yes',
       'housing_no', 'housing_unknown', 'housing_yes', 'loan_no',
       'loan_unknown', 'loan_yes', 'contact_cellular', 'contact_telephone',
       'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu',
       'day_of_week_tue', 'day_of_week_wed', 'poutcome_failure',
       'poutcome_nonexistent', 'poutcome_success'],
      dtype='object')
```

Figure 3.42: Resulting features, which will be used in the logistic regression model

2. Transform the target variable:

```
    # extract and transform target variable
    y = np.where(bank_data["y"] == "yes", 1 ,0)
```

3. Finally, create a logistic regression model based on the full set of features:

```
    # define and fit model
    full_logistic_regression_model = sm.Logit(y, X)
    result = full_logistic_regression_model.fit(maxiter=500)
    print(result.summary())
```

The result of the model is shown in the following figure:

Logit Regression Results

Dep. Variable:	y	No. Observations:	41188
Model:	Logit	Df Residuals:	41135
Method:	MLE	Df Model:	52
Date:	Wed, 12 Feb 2020	Pseudo R-squ.:	0.4111
Time:	17:31:02	Log-Likelihood:	-8538.9
converged:	False	LL-Null:	-14499.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-68.6994	nan	nan	nan	nan	nan
age	0.0002	0.002	0.081	0.936	-0.005	0.005
duration	0.0047	7.46e-05	63.108	0.000	0.005	0.005
campaign	-0.0401	0.012	-3.473	0.001	-0.063	-0.017
pdays	-0.0009	0.000	-4.326	0.000	-0.001	-0.001
previous	-0.0628	0.059	-1.062	0.288	-0.179	0.053
emp.var.rate	-1.7576	0.142	-12.380	0.000	-2.036	-1.479
cons.price.idx	2.1905	0.252	8.679	0.000	1.696	2.685
cons.conf.idx	0.0207	0.008	2.664	0.008	0.005	0.036
euribor3m	0.3316	0.130	2.551	0.011	0.077	0.586
nr.employed	0.0054	0.003	1.738	0.082	-0.001	0.012
job_admin.	-5.5134	2.11e+06	-2.62e-06	1.000	-4.13e+06	4.13e+06
job_blue-collar	-5.7481	1.82e+06	-3.16e-06	1.000	-3.56e+06	3.56e+06
job_entrepreneur	-5.6914	1.77e+06	-3.22e-06	1.000	-3.46e+06	3.46e+06
job_housemaid	-5.5377	nan	nan	nan	nan	nan
job_management	-5.5695	1.9e+06	-2.92e-06	1.000	-3.73e+06	3.73e+06
job_retired	-5.2276	1.37e+06	-3.8e-06	1.000	-2.69e+06	2.69e+06
job_self-employed	-5.6712	1.69e+06	-3.36e-06	1.000	-3.3e+06	3.3e+06
job_services	-5.6532	2.42e+06	-2.34e-06	1.000	-4.74e+06	4.74e+06
job_student	-5.3100	1.93e+06	-2.76e-06	1.000	-3.77e+06	3.77e+06
job_technician	-5.5274	1.9e+06	-2.9e-06	1.000	-3.73e+06	3.73e+06
job_unemployed	-5.4922	1.38e+06	-3.97e-06	1.000	-2.71e+06	2.71e+06
job_unknown	-5.5838	2.55e+06	-2.19e-06	1.000	-5.01e+06	5.01e+06
marital_divorced	-16.6541	9.63e+05	-1.73e-05	1.000	-1.89e+06	1.89e+06
marital_married	-16.6569	3.31e+05	-5.04e-05	1.000	-6.48e+05	6.48e+05
marital_single	-16.5982	3.1e+05	-5.36e-05	1.000	-6.07e+05	6.07e+05
marital_unknown	-16.6247	7.8e+05	-2.13e-05	1.000	-1.53e+06	1.53e+06
education_basic.4y	-8.5265	1.47e+06	-5.79e-06	1.000	-2.89e+06	2.89e+06
education_basic.6y	-8.4039	1.46e+06	-5.77e-06	1.000	-2.86e+06	2.86e+06
education_basic.9y	-8.5275	1.44e+06	-5.92e-06	1.000	-2.82e+06	2.82e+06
education_high.school	-8.4779	1.46e+06	-5.83e-06	1.000	-2.85e+06	2.85e+06
education_illiterate	-7.4598	1.45e+06	-5.14e-06	1.000	-2.84e+06	2.84e+06
education_professional.course	-8.4115	1.47e+06	-5.72e-06	1.000	-2.88e+06	2.88e+06
education_university.degree	-8.3306	1.44e+06	-5.77e-06	1.000	-2.83e+06	2.83e+06
education_unknown	-8.3768	1.44e+06	-5.83e-06	1.000	-2.82e+06	2.82e+06
default_no	-16.6776	nan	nan	nan	nan	nan
default_unknown	-16.9779	nan	nan	nan	nan	nan
default_yes	-32.8750	nan	nan	nan	nan	nan
housing_no	-22.1674	nan	nan	nan	nan	nan
housing_unknown	-22.2021	nan	nan	nan	nan	nan
housing_yes	-22.1721	nan	nan	nan	nan	nan
loan_no	-22.1438	3.57e+05	-6.2e-05	1.000	-7e+05	7e+05
loan_unknown	-22.2024	nan	nan	nan	nan	nan
loan_yes	-22.1954	3.73e+05	-5.95e-05	1.000	-7.32e+05	7.32e+05
contact_cellular	-32.9450	nan	nan	nan	nan	nan
contact_telephone	-33.5910	nan	nan	nan	nan	nan
month_apr	-6.9063	1.54e+06	-4.5e-06	1.000	-3.01e+06	3.01e+06
month_aug	-6.0410	1.53e+06	-3.94e-06	1.000	-3e+06	3e+06
month_dec	-6.5871	1.53e+06	-4.32e-06	1.000	-2.99e+06	2.99e+06
month_jul	-6.7717	1.5e+06	-4.51e-06	1.000	-2.94e+06	2.94e+06
month_jun	-7.4306	1.41e+06	-5.26e-06	1.000	-2.77e+06	2.77e+06
month_mar	-4.8922	1.52e+06	-3.22e-06	1.000	-2.98e+06	2.98e+06
month_may	-7.3502	1.5e+06	-4.9e-06	1.000	-2.94e+06	2.94e+06
month_nov	-7.3243	1.48e+06	-4.95e-06	1.000	-2.9e+06	2.9e+06
month_oct	-6.7123	1.46e+06	-4.59e-06	1.000	-2.86e+06	2.86e+06
month_sep	-6.5323	1.47e+06	-4.44e-06	1.000	-2.88e+06	2.88e+06
day_of_week_fri	-13.4901	nan	nan	nan	nan	nan
day_of_week_mon	-13.6069	nan	nan	nan	nan	nan
day_of_week_thu	-13.4341	nan	nan	nan	nan	nan
day_of_week_tue	-13.3929	nan	nan	nan	nan	nan
day_of_week_wed	-13.3148	nan	nan	nan	nan	nan
poutcome_failure	-22.8763	nan	nan	nan	nan	nan
poutcome_nonexistent	-22.4505	nan	nan	nan	nan	nan
poutcome_success	-21.9166	nan	nan	nan	nan	nan

Figure 3.43: Result of the logistic regression model when run on the full set of features

As shown in the preceding figure, the model we obtained is quite large. In fact, we have 64 different features (after encoding the categorical variables as dummy ones). It would be quite optimistic to expect that all the obtained features have a significant impact on our model. Also, a model with so many parameters would be difficult to interpret.

> **NOTE**
>
> To access the source code for this specific section, please refer to https://packt.live/3hwyUgU.
>
> You can also run this example online at https://packt.live/2AwGGH9. You must execute the entire Notebook in order to get the desired result.

In fact, observing the `P>|z|` column in the preceding figure shows us that for most of the features, the p-value is equal to one, which means that we cannot reject the null hypothesis, stating that the coefficient for the respective column should be equal to zero. In other words, we have lots of redundant columns in our model.

Furthermore, for some of the coefficients, we cannot even compute the p-values (in fact, there is a lot of NaN present in the result, indicating that the optimization model running on the back of the `Logit.fit()` function is not able to converge, which is another reason to reduce the number of features).

We will deal with this issue in the next activity. For now, it is important to remember that even if we might be able to achieve a higher accuracy model with more features, it is not always the best choice, especially if we are trying to explain the impact the different features have on our target variable.

Let's perform an activity to create a more detailed logistic regression model.

ACTIVITY 3.01: CREATING A LEANER LOGISTIC REGRESSION MODEL

The aim of this activity is to create a leaner logistic regression model by accurately selecting the important features and, in this way, improve the result of the model from *Exercise 3.04, Logistic Regression on the Full Marketing Campaign Data.* In this way, you will obtain an explainable logistic regression model with far fewer features to be combined.

It is always important to use as few features as possible when training linear models as one of their greatest advantages is their interpretability. By the end of this activity, you should be confident in building logistic regression models, improving their results, and interpreting the values of the different parameters from a business perspective.

The following steps should help you complete this activity:

1. Import the necessary Python packages.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
```

2. Upload the **`bank-additional-full.csv`** dataset from the data folder on GitHub.
3. Create a feature matrix using a selection of the variables with a p-value smaller than 0.05 (see Figure 3.43 for the appropriate p-values). See the table below for a list of variables to include:

   The output will be as follows:

	const	duration	campaign	pdays	cons.price.idx	cons.conf.idx	euribor3m
0	1.0	261	1	999	93.994	-36.4	4.857
1	1.0	149	1	999	93.994	-36.4	4.857
2	1.0	226	1	999	93.994	-36.4	4.857
3	1.0	151	1	999	93.994	-36.4	4.857
4	1.0	307	1	999	93.994	-36.4	4.857

Figure 3.44: Feature matrix to be used in the logistic regression model

4. Transform the target variable into an array containing **0** and **1** (where 0 corresponds to "no" entries in the **`y`** column).