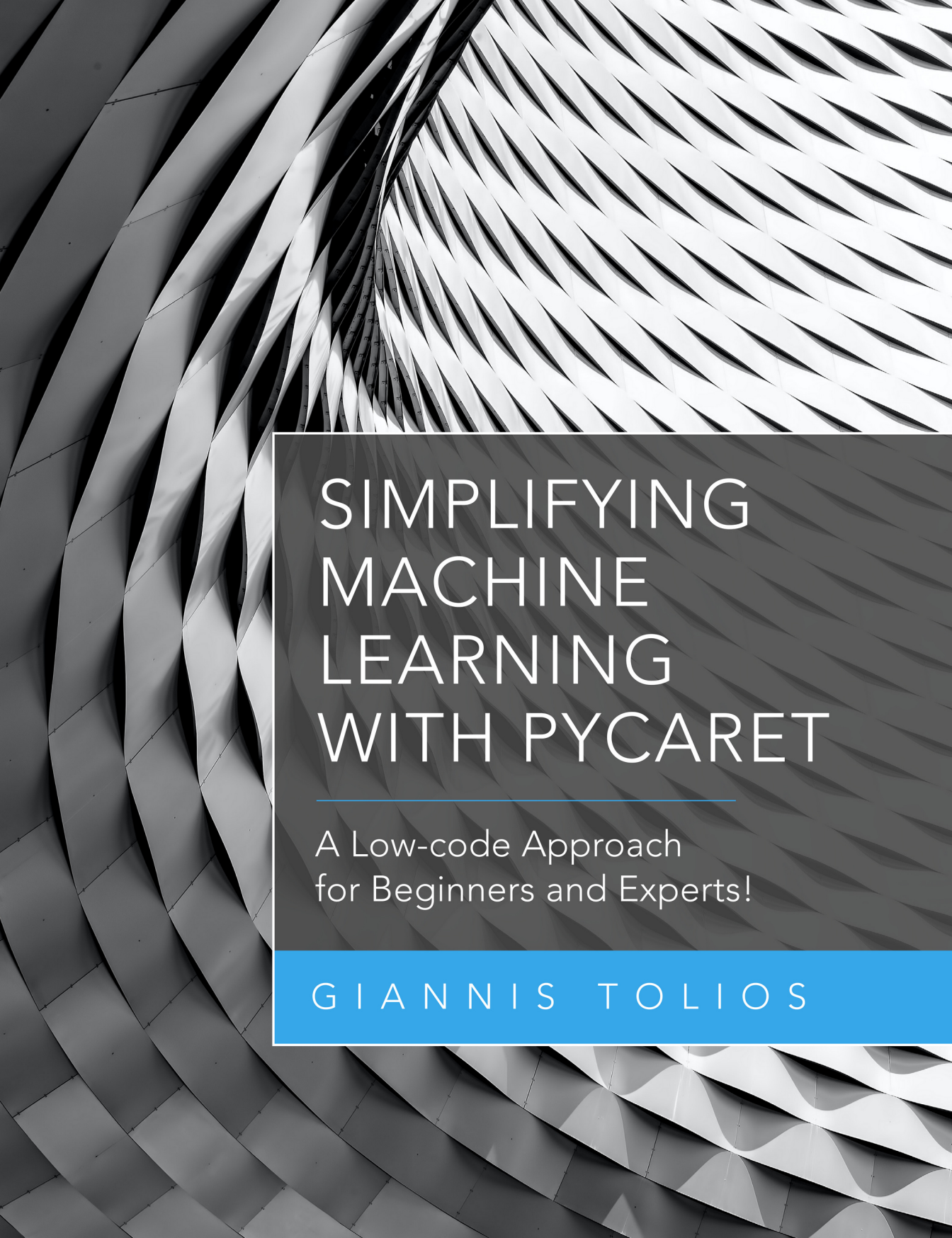


SIMPLIFYING MACHINE LEARNING WITH PYCARET

A Low-code Approach for Beginners and Experts!

GIANNIS TOLIOS

Simplifying Machine Learning with PyCaret

A Low-code Approach for Beginners and Experts!

Giannis Tolios

This book is for sale at http://leanpub.com/pycaretbook

This version was published on 2022-01-19

Contents

Preface

I decided to write this book for a couple of reasons. After publishing articles on websites like Towards Data Science, I was interested in taking a step forward as a writer. In the beginning of 2021, a major international publisher proposed me to collaborate on a PyCaret book, as I had already written an article about that library. At first I was interested, but eventually decided to publish my book on Leanpub, as it made more sense from a financial perspective. Furthermore, at the time of writing there is no other PyCaret book available, while there seems to be significant demand for it.

In this book, I will present an overview of all the main features of PyCaret, by focusing on practical case studies that are easy to follow. After reading it, you'll be able to create and deploy machine learning models, thus taking advantage of this powerful technology. Machine learning is a field that has grown substantially in the past years, due to technological and scientific advancements. Data scientists and machine learning engineers are among the best paid professionals in the modern job market, so learning those skills is beneficial for every developer. The low-code approach of PyCaret makes it suitable for beginners, as well as seasoned developers that want to get results quickly, thus increasing their productivity.

The following chapters contain hands-on tutorials for the main machine learning modules that are included with PyCaret, such as regression, classification, anomaly detection, clustering and natural language processing. Furthermore, I will also explain how to deploy ML models as web applications, based on the Streamlit library and the associated Streamlit Cloud service. This will let you share your work with others, thus showcasing your skill and expertise. I hope you enjoy reading this book, and I encourage you to share your thoughts and feedback[1] with me!

[1] https://forms.gle/1hsbBtG1ZSEcyrH27

About the Author

Giannis Tolios is a data scientist who is passionate about expanding his knowledge and evolving as a professional. He has collaborated with numerous companies worldwide as a freelancer, and completed projects related to machine learning, NLP, time series forecasting, scientific data visualization and others. Giannis also enjoys writing about data science[2] at established websites such as Towards Data Science and Analytics Vidhya. Giannis strongly believes that technology should be used for good, and is constantly looking for new ways to help mitigate challenges like climate change and economic inequality, by using data science. If you want to learn more about Giannis, you can visit his personal website[3], or follow him on LinkedIn[4], where he's regularly posting content about data science and other topics.

[2] https://giannistolios.medium.com/
[3] https://giannis.io
[4] https://www.linkedin.com/in/giannis-tolios

About the Book

Who this Book is for

This book is targeted towards Python developers that want to familiarize themselves with the PyCaret library, as well as machine learning in general. The content is beginner-friendly, so I assume no prior knowledge of machine learning techniques and methods. Experienced data scientists and machine learning engineers can also benefit from reading this book, as it will help them acquaint themselves with the low-code approach of PyCaret, and improve their workflow. This book will focus on practical coding examples, so I will only provide basic explanations of theoretical concepts. In case you want to dig deeper into the theory, I suggest that you start with An Introduction to Statistical Learning[5], an excellent book that is provided as a free PDF download by the authors.

Prerequisites

You should have some basic knowledge of Python 3, linear algebra, statistics and probability theory. The math included in this book will be minimal, as it isn't supposed to be a college textbook, but rather a practical guide for developers. Regardless, having a solid understanding of some fundamental mathematical concepts is important. The following resources should be helpful to beginners and people who want to freshen up their math skills.

- The Official Python Tutorial[6]
- Free Linear Algebra Course on Khan Academy[7]
- Free Statistics and Probability Course on Khan Academy[8]

[5] https://www.statlearning.com/
[6] https://docs.python.org/3/tutorial/index.html
[7] https://www.khanacademy.org/math/linear-algebra
[8] https://www.khanacademy.org/math/statistics-probability/

Software Requirements

The code in this book should work on all major operating systems, i.e. Microsoft Windows, Linux and Apple macOS. You will need to have Python 3 installed on your computer, as well as JupyterLab. I suggest that you use Anaconda[9], a machine learning and data science toolkit that includes numerous helpful libraries and software packages. Anaconda can be freely downloaded at this link[10]. Alternatively, you can use a cloud service like Google Colab[11] to run Python code without worrying about installing anything on your machine.

Installing PyCaret

The PyCaret library can be installed locally by executing the following command on your Anaconda terminal. You can also execute the same command on Google Colab or a similar service, to install the library on a remote server.

```
pip install pycaret[full]==2.3.4
```

After executing this command, PyCaret will be installed and you'll be able to run all code examples of the book. It is recommended to install the optional dependencies as well, by including the `[full]` specifier. Furthermore, installing the correct package version ensures maximum compatibility, as I used PyCaret ver. 2.3.4 while working on this book. Finally, creating a conda environment[12] for PyCaret is considered to be best practice, as it will help you avoid conflicts with other packages and make sure you always have the correct dependencies installed.

[9] https://www.anaconda.com/
[10] https://www.anaconda.com/products/individual
[11] https://colab.research.google.com/
[12] https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html#managing-envs

Using JupyterLab

JupyterLab is a powerful web-based user interface that lets you create and execute Jupyter notebooks containing Python code. You can run JupyterLab by executing the following command on the Anaconda terminal. Substituting `lab` with `notebook` will run the older Jupyter Notebook web interface, but I only recommend it if you have problems or compatibility issues with JupyterLab.

```
jupyter lab
```

When JupyterLab starts, you can create a new Jupyter notebook and run the code that is provided in each chapter of the book. After executing a Jupyter notebook, both the code and its output are displayed to the user. This file format can also contain text, figures and visualizations. Jupyter notebook is a powerful format that has become the standard in data science and machine learning. You can refer to the official JupyterLab Documentation[13] for more information and help about using JupyterLab. In the following chapters, each code snippet represents a single Jupyter notebook cell that can be executed by pressing the Shift + Enter keyboard shortcut.

Github Repository

You can access all code examples included in this book at the official Github repository[14], with each folder containing the Python code of the same-titled book chapter. You can also clone the repository to your local machine if you wish. Git is the de facto standard for version control, so I assume that most developers will be familiar with it. In case you haven't used it, please refer to the official quick reference guide[15].

[13] https://jupyterlab.readthedocs.io/en/stable/index.html
[14] https://github.com/derevirn/pycaret-book
[15] https://training.github.com/downloads/github-git-cheat-sheet/

1. A Brief Introduction to Machine Learning

Before starting a machine learning project, you need to have a solid grasp of the theory behind that technology. This book is targeted mostly to beginners and Python developers that want to improve their skill set and familiarize themselves with PyCaret, so I assume no prior knowledge of machine learning. Therefore, I will provide a brief introduction to the basic theoretical concepts that are necessary to understand the rest of the book. In case you are familiar with them, feel free to skip this chapter.

What is Machine Learning?

The term machine learning was popularized by Arthur Samuel, a prominent figure in the field of artificial intelligence[1]. Samuel defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. This was a revolutionary approach, as traditional computer algorithms include all the necessary steps that have to be executed, explicitly defined by a human programmer. Unfortunately though, some problems are simply too difficult and complex, making it almost impossible to define an algorithm for them. Machine learning can be used to circumvent that obstacle by training the computer to find the solution, rather than giving it a set of explicit instructions. You may see machine learning being used interchangeably with the terms artificial intelligence and deep learning. Although associated, those terms aren't identical. It is generally accepted that machine learning is a subfield of artificial intelligence, while deep learning is a subfield of machine learning.

Machine Learning Categories

Machine learning is typically divided in three main categories, i.e. supervised learning, unsupervised learning and reinforcement learning[2]. I will give a brief description for each one of those categories, as well as some real-life examples so you can easily get a grasp of them.

Supervised Learning

The main goal in supervised learning is to create a predictive model based on a set of labeled instances, known as the training dataset[3]. This dataset is a matrix where rows represent the instances, while columns represent the features and label. After the model is successfully trained, it can be used to make predictions on unlabeled data. Supervised learning is the most common type of machine learning, including tasks such as regression and classification. It can be applied to various kinds of data, like structured/tabular, text, image and video.

To help you understand supervised learning better, I am going to describe a simple regression example, where the goal is to predict a continuous value. In this case, the training dataset includes cars with attributes like dimensions, horsepower, engine size, number of cylinders and gas mileage. Those attributes are going to be used as features, while price is considered to be the label. After training a regression model on that dataset, it will be able to estimate the price of a car based on the rest of its attributes. We are going to examine a complete case study of regression in the next chapter.

Let's also consider a classification example, where the goal is to predict a categorical class label. The dataset is comprised of patient records, including attributes such as age, sex, resting blood pressure, chest pain type and maximum heart rate. Those attributes are the features, while the presence or absence of heart disease in each patient is the label. This is known as a binary classification task, because there are only two classes. The trained model, known as a classifier in this case, will be able to evaluate whether a patient is at risk of having heart disease. Classification will be covered in chapter 3 of this book.

Unsupervised Learning

In case we can't get a dataset with labeled instances, unsupervised learning algorithms can help us discover hidden structures on our data. Unsupervised learning includes tasks such as clustering and anomaly detection. Those algorithms can be applied to structured data, text, image and other types of data, similarly to supervised learning.

Let's consider the following clustering example. Suppose that a company has a dataset with information about its customers, including various attributes, such as age, income and spending behavior. Groups of customers with similar characteristics can be created with the help of a clustering algorithm, in a process known as market segmentation. That information can be used to create effective marketing campaigns for each group, thus improving company sales and increasing revenue. We are going to examine a clustering case study in chapter 4.

Anomaly detection can be used in numerous different cases, such as fraud in financial transactions. In this case, every transaction is analyzed by an anomaly detection algorithm to detect potential fraud cases. Fraudulent transactions may include various possible events, like stolen credit cards being used for purchases, or hackers attempting to transfer funds illegally. Every suspicious transaction will be flagged as such by the algorithm. Anomaly detection will be studied in chapter 5.

Reinforcement Learning

The goal of reinforcement learning is to create an intelligent agent that interacts with its environment. That agent gets rewarded for every correct decision it makes, thus learning to perform various actions via trial-and-error[4]. Reinforcement learning can be used to create autonomous vehicles such as self-driving cars[5]. It can also be used to improve the AI of video games, thus resulting in more immersive and realistic experiences for gamers. We are not going to examine reinforcement learning in detail, as PyCaret doesn't support it currently.

2. Regression

The Linear Regression Model

A fundamental task in supervised machine learning is regression, where the goal is to predict a continuous value. This is achieved by understanding the relationship between the target variable y and the feature variables x on a given dataset. One of the most basic regression models is Linear Regression[6], which is defined in the following equation. The equivalent vectorized form of the equation is also provided, where the inner product of the transposed vector $\boldsymbol{\beta}^\top$ and $\boldsymbol{X_n}$ is calculated.

$$y_n = \beta_0 + \beta_1 x_{n1} + \cdots + \beta_p x_{np} + \epsilon_n = \boldsymbol{\beta}^\top \boldsymbol{X_n} + \epsilon_n$$

- y_n is the target variable for the n^{th} instance of the given dataset.
- x_1 to x_p are the feature variables.
- β_0 is the intercept term.
- β_1 to β_p are the coefficients of the feature variables.
- ϵ is the error variable.

Regression with PyCaret

Besides Linear Regression, there are numerous other models available, such as Lasso[7], Random Forest[8], Support Vector Machines[9] and Gradient Boosting[10]. In the rest of this chapter, we are going to see how PyCaret can help us choose and train the optimal regression model for a specific dataset. We are also going to learn about Exploratory Data Analysis (EDA), a method that lets you examine and understand the basic statistical properties of a dataset.

Importing the Necessary Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from pycaret.datasets import get_data
from pycaret.regression import *
mpl.rcParams['figure.dpi'] = 300
```

First of all, we import the Python libraries that are necessary for our project. Some standard machine learning libraries are included, such as pandas, Matplotlib and Seaborn. Furthermore, we import all PyCaret functions that are related to regression. The last line specifies that Matplotlib figures will have a 300 DPI resolution, but you can omit that if you wish.

Loading the Dataset

Machine learning projects can only succeed if the proper data are available, so PyCaret includes a variety of datasets that can be used to test its features. In this chapter, we are going to use `insurance.csv`, a dataset that originates from the book Machine Learning with R, by Brett Lantz[11]. This is a health insurance dataset, where the features are various attributes including age, sex, Body Mass Index (BMI)[1], whether the person is a smoker or not, number of children and US region. Furthermore, the dataset target variable is the billed charges for every individual. Real-world data are usually more complex, but working with so-called toy datasets will help you grasp the concepts and techniques before dealing with more difficult cases.

[1] https://en.wikipedia.org/wiki/Body_mass_index

```
data = get_data('insurance')
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

We use the `get_data()` PyCaret function to load the dataset to a pandas dataframe. The output is equivalent to the `head()` pandas function that prints the first 5 dataset rows. This lets us get a first glimpse of the data we are working with.

```
data.info()
```

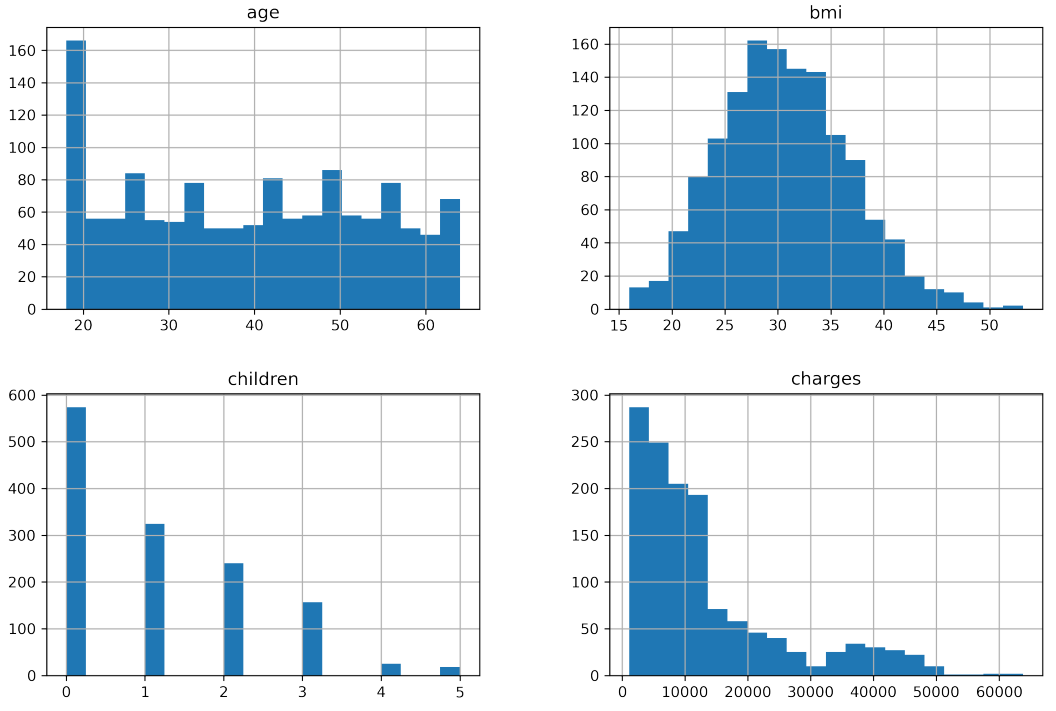
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

We use the pandas `info()` function to examine some basic information about the dataset. As we can see, there are 1338 rows and none of the columns have null values. Furthermore, the data type of each column has been automatically inferred by the pandas library.

Exploratory Data Analysis

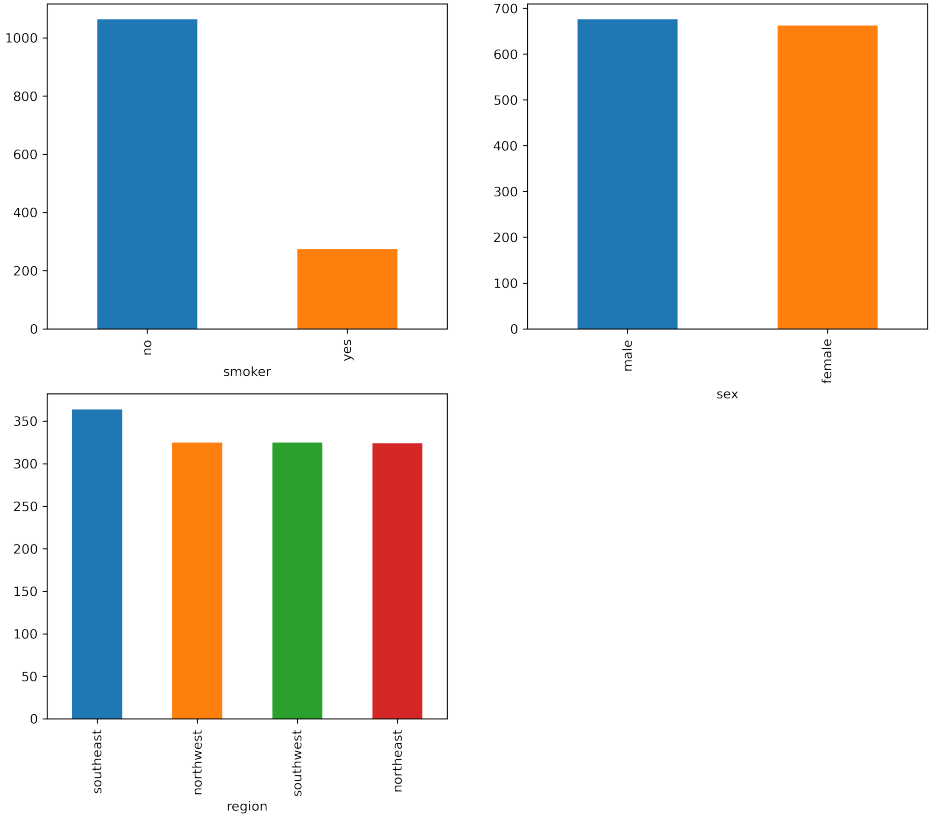
We are now going to perform Exploratory Data Analysis (EDA) on our data. As mentioned earlier, EDA is a method that helps us understand the dataset properties by using descriptive statistics and visualization. It is an important part of every machine learning or data science project, as you have to understand the dataset before being able to utilize it.

```
numeric = ['age', 'bmi', 'children', 'charges']

data[numeric].hist(bins=20, figsize = (12,8))
plt.show()
```



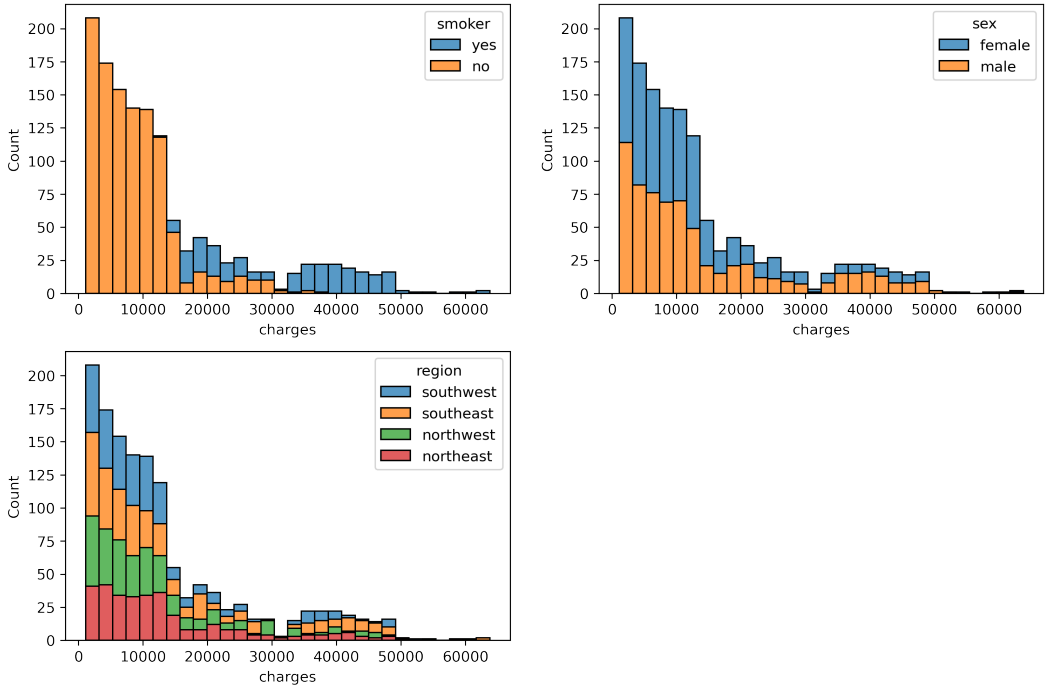
The distribution of numeric variables can be visualized with a histogram that can be easily created with the `hist()` pandas function. It is obvious that some of the variables have right-skewed distributions that may cause problems with regression models, so we'll have to deal with that later.

```
categorical = ['smoker', 'sex', 'region']
color = ['C0', 'C1', 'C2', 'C3']
fig, axes = plt.subplots(2, 2, figsize = (12,10))
axes[1,1].set_axis_off()

for ax, col in zip(axes.flatten(), categorical) :
    data[col].value_counts().plot(kind = 'bar', ax = ax, color = color)
    ax.set_xlabel(col)
```



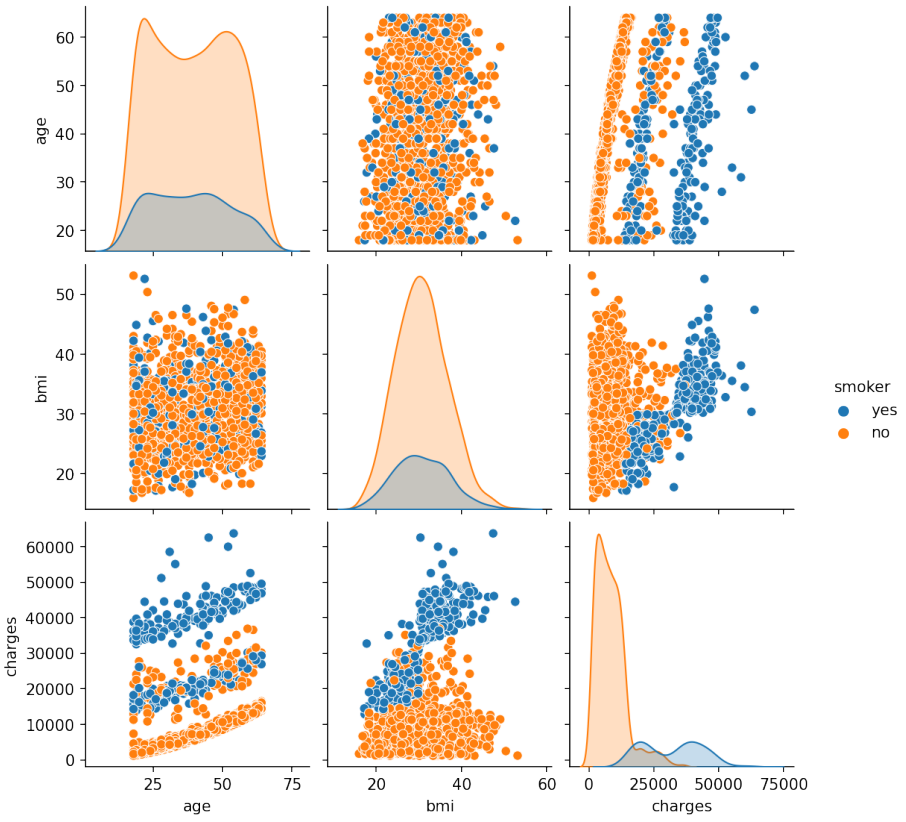
Using bar charts is the standard way of plotting categorical variables. We can accomplish that easily, by using the `value_counts()` and `plot()` pandas functions. As we can see, the smoker variable has uneven distribution, with only 20% of people being smokers. On the other hand, the sex and region variables are equally distributed.

```
fig, axes = plt.subplots(2, 2, figsize=(12,8))
axes[1,1].set_axis_off()

for ax, col in zip(axes.flatten(), categorical):
    sns.histplot(data, x='charges', hue=col, multiple='stack', ax=ax)
```



The `histplot()` Seaborn function lets us visualize the relationship between numeric and categorical variables by using hue mapping. In this case, we plot the target variable histogram, colored differently for every category of the smoker, sex and region variables. Evidently, smokers get significantly higher charges compared to non-smokers. This is expected, as the health risks associated with smoking are numerous and well-documented.

```
cols = ['age', 'bmi', 'charges', 'smoker']
sns.pairplot(data[cols], hue='smoker')
plt.show()
```



Scatter plots are a type of visualization that helps us understand the relationship between numeric variables. The `pairplot()` Seaborn function creates a grid of scatter plots for all pairs of numeric variables in a given dataset. The diagonal contains distribution plots of the variables, such as histograms or KDE plots in this case. Once again, we use hue mapping to highlight the differences between smokers and non-smokers. As we can see, age is correlated with charges, i.e. people get higher charges as they grow older. In spite of that, being a non-smoker keeps the cost lower for most people, regardless of their age. Furthermore, overweight and obese people don't seem to get significantly higher charges, unless they are also smokers.

Initializing the PyCaret Environment

```
reg = setup(data=data, target='charges', train_size = 0.8, session_id = 7402,
            numeric_features = numeric[:-1], categorical_features = categorical,
            transformation = True, normalize = True, transform_target = True)
```

	Description	Value
0	session_id	7402
1	Target	charges
2	Original Data	(1338, 7)
3	Missing Values	False
4	Numeric Features	3
5	Categorical Features	3
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(1070, 9)
10	Transformed Test Set	(268, 9)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold

After the EDA part of the project is complete, the next step is to initialize the PyCaret environment. We can accomplish that by using the `setup()` function, which prepares the data for model training and deployment. This function has numerous parameters, but we are only going to focus on the most important. In case you want to delve deeper, you can refer to the documentation page of the PyCaret Regression module[2].

[2] https://pycaret.readthedocs.io/en/latest/api/regression.html

After running the `setup()` function, a table with its parameters and settings is printed, as seen in the image. We are now going to examine the preprocessing pipeline that has been applied to the dataset.

Identifying Numeric and Categorical Features

PyCaret can automatically infer whether a feature is numeric or categorical. When you execute the `setup()` function, you will be prompted to verify that the features have been identified correctly. Alternatively, you can specify which features are categorical or numeric using the `categorical_features` and `numeric_features` parameters, as we did in this case.

Train/Test Split

Splitting a dataset to a train and test subset is standard practice in machine learning, because it is important to evaluate model performance on data that haven't been used to train it. In this case, we have set the `train_size` parameter to 0.8, meaning that the machine learning model will be trained on 80% of the original data, while the 20% will be used for testing purposes.

Normalization of Numeric Features

Some regression models require numeric features to be normalized by having $\mu = 0$ and $\sigma = 1$. We enabled normalization by setting the `normalize` parameter to `True`, thus applying the standard scaler. This method replaces each value of the feature with the z-score, which is defined as $z = \frac{x-\mu}{\sigma}$. Changing the `normalize_method` parameter lets us choose a different normalization method, with other options including the Min Max scaler and the Robust scaler.

Transformation of Numeric Features and Target

In the EDA part of the project, we realized that some of the features and the target variable are skewed. This can degrade the performance of some regression models, as they are designed to work with variables that have a normal distribution. We can deal with that problem by transforming the variables so their distribution is closer to normal. Transformation was enabled by setting the `transformation` and `transform_target` parameters to `True`.

One-Hot Encoding Categorical Features

Categorical features are supported by some regression models, but they can cause problems with others. It is therefore advised to apply a categorical encoding method, thus converting them to numeric variables. PyCaret applies one-hot encoding by default, where new binary features are created for each category of the categorical features.

Printing the Preprocessed Features

```
get_config('X').head()
```

	age	bmi	children	sex_female	smoker_no	region_northeast	region_northwest	region_southeast	region_southwest
0	-1.462763	-0.408971	-1.053884	1.0	0.0	0.0	0.0	0.0	1.0
1	-1.535749	0.543953	0.203960	0.0	1.0	0.0	0.0	1.0	0.0
2	-0.807782	0.426243	1.414612	0.0	1.0	0.0	0.0	1.0	0.0
3	-0.445654	-1.350171	-1.053884	0.0	1.0	0.0	1.0	0.0	0.0
4	-0.517962	-0.240519	-1.053884	0.0	1.0	0.0	1.0	0.0	0.0

This is what the feature variables look after the preprocessing pipeline has been applied to them. We can see that the numeric features have different values due to the normalization/transformation, and the categorical features have been converted to multiple binary variables due to the one-hot encoding.

Comparing Regression Models

```
best = compare_models(sort='RMSE')
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2275.4641	22815428.3119	4750.1089	0.8350	0.3858	0.1874	0.0190
rf	Random Forest Regressor	2342.1429	22959433.7357	4762.0337	0.8351	0.4097	0.2091	0.0760
ada	AdaBoost Regressor	3257.2171	23279230.3521	4807.0257	0.8339	0.4770	0.4264	0.0100
lightgbm	Light Gradient Boosting Machine	2491.6919	24030584.9610	4865.2118	0.8272	0.4153	0.2118	0.0590
catboost	CatBoost Regressor	2485.6847	25075332.3085	4978.9614	0.8189	0.4071	0.1994	0.5830
et	Extra Trees Regressor	2364.4206	25237906.1980	4999.4284	0.8167	0.4283	0.2116	0.0680
xgboost	Extreme Gradient Boosting	2931.6919	31946244.2000	5615.7612	0.7678	0.4551	0.2602	0.0920
dt	Decision Tree Regressor	3031.4152	42283353.7664	6468.0098	0.6936	0.5132	0.3181	0.0090
omp	Orthogonal Matching Pursuit	5645.3006	59119658.9118	7679.3608	0.5758	0.6831	0.6880	0.0080
ridge	Ridge Regression	4066.3602	61583198.0000	7714.4266	0.5583	0.4400	0.2707	0.0080
br	Bayesian Ridge	4072.9367	61948316.9816	7735.3075	0.5556	0.4399	0.2705	0.0080
lar	Least Angle Regression	4081.2537	62419165.3459	7762.5119	0.5521	0.4399	0.2702	0.0080
lr	Linear Regression	4081.2544	62419200.4000	7762.5147	0.5521	0.4399	0.2702	0.0080
knn	K Neighbors Regressor	4590.2544	70154126.3724	8271.6145	0.5162	0.5252	0.3131	0.0200
huber	Huber Regressor	4211.3096	80449305.4129	8799.2293	0.4214	0.4535	0.2076	0.0090
par	Passive Aggressive Regressor	5841.9890	94762106.9683	9607.9039	0.2863	0.6406	0.4609	0.0090
en	Elastic Net	8222.6688	160918303.2000	12608.0390	-0.1206	0.9079	0.9707	0.0080
llar	Lasso Least Angle Regression	8249.2145	161224210.7582	12619.7361	-0.1227	0.9107	0.9777	0.0080
lasso	Lasso Regression	8249.2145	161224220.8000	12619.7366	-0.1227	0.9107	0.9777	0.0080

As we mentioned earlier, there are numerous regression models available, and choosing the best one for our data can be challenging. The `compare_models()` function simplifies this process, by training all the available models and displaying a table with the results. The first column contains the regression model name, while the rest are various metrics. This table may seem intimidating if you are a beginner, but it actually isn't that complicated. As we can see, there are various metrics that can be used to evaluate the performance of a regression model, but we are going to focus on Root Mean Squared Error (RMSE), so we sorted the results based on it.

RMSE is one of the most widely used metrics for regression, and it is defined as the square root of the average of squared errors, between actual and predicted values. Lower RMSE values indicate a more accurate regression model, so in this case, the most accurate model is Gradient Boosting Regressor with an RMSE value of 4750.1089.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$$

Creating the Model

```
model = create_model('gbr', fold = 10)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2324.8131	24607040.5795	4960.5484	0.8393	0.3810	0.1751
1	2571.2202	29114809.8137	5395.8141	0.7413	0.4376	0.2551
2	2483.0865	21498482.3051	4636.6456	0.8585	0.4261	0.2000
3	2545.2911	25489822.2086	5048.7446	0.8158	0.3937	0.1765
4	2218.7259	19926308.9440	4463.8894	0.8240	0.4237	0.2161
5	1991.9530	17546103.1066	4188.8069	0.9050	0.2883	0.1446
6	2557.6356	30138770.5115	5489.8789	0.8316	0.4120	0.1911
7	1904.3245	16102540.3261	4012.7971	0.8950	0.3101	0.1860
8	1789.5416	17459155.1593	4178.4154	0.8269	0.3299	0.1574
9	2368.0500	26271250.1647	5125.5488	0.8125	0.4560	0.1721
Mean	2275.4641	22815428.3119	4750.1089	0.8350	0.3858	0.1874
SD	274.1307	4766740.8468	501.8900	0.0435	0.0546	0.0297

We can now train the Gradient Boosting Regressor by using the `create_model()` function, which uses k-fold cross validation to evaluate model accuracy. The dataset is partitioned into k subsamples, with one of them being retained for validation, while the rest are used to train the model. This process is repeated k times, with the resulting metrics for each subsample being displayed when it's completed.

Tuning the Model

```
params = {'learning_rate': [0.01, 0.02, 0.05],
        'max_depth': [1,2, 3, 4, 5, 6, 7, 8],
        'subsample': [0.4, 0.5, 0.6, 0.7, 0.8],
        'n_estimators' : [100, 200, 300, 400, 500, 600]}

tuned_model = tune_model(model, optimize = 'RMSE', fold = 10,
                         custom_grid = params, n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2221.3251	23485165.0834	4846.1495	0.8466	0.3680	0.1698
1	2413.4985	28417256.3250	5330.7838	0.7475	0.4269	0.2226
2	2358.1966	20434514.3855	4520.4551	0.8655	0.4078	0.1862
3	2224.8093	22543132.0952	4747.9608	0.8371	0.3895	0.1590
4	1954.8899	17295001.5227	4158.7259	0.8472	0.4099	0.1875
5	1866.4875	16659066.2449	4081.5519	0.9098	0.2809	0.1412
6	2486.6686	29110527.1669	5395.4172	0.8374	0.4152	0.1857
7	1862.5828	15867899.1150	3983.4532	0.8966	0.2974	0.1748
8	1746.2978	16452845.2892	4056.2107	0.8368	0.3261	0.1598
9	2372.9397	26292453.2888	5127.6167	0.8124	0.4505	0.1706
Mean	2150.7696	21655786.0517	4624.8325	0.8437	0.3772	0.1757
SD	255.0557	4826683.3525	516.4401	0.0425	0.0546	0.0208

Hyperparameter tuning is a technique where the various parameters of a machine learning model are tweaked to optimize its performance. The `tune_model()` function can be used to apply hyperparameter tuning on a trained model by using the randomized search method, where a random sample of possible combinations is tested.

A dictionary with the hyperparameters of our preference was passed to the `custom_grid` parameter of the function. We can see that after tuning the model, the RMSE value decreased to 4624.8325. You should keep in mind that hyperparameter tuning can be a computationally intensive and time-consuming process. In case it takes too long to complete on your computer, you can try decreasing the number of iterations by setting a lower value on the `n_iter` parameter. Each regression model has a different set of hyperparameters that can be tuned, so you'll have to consult its documentation to select them. In this case, you can refer to the Gradient Boosting Regression[3] documentation page of scikit-learn[4].

Making Predictions

```
cols = ['age', 'bmi', 'children', 'sex_female', 'smoker_no','charges','Label']
predictions = predict_model(tuned_model)
predictions[cols].head()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	1937.1153	16816546.1060	4100.7982	0.8921	0.3297	0.1730

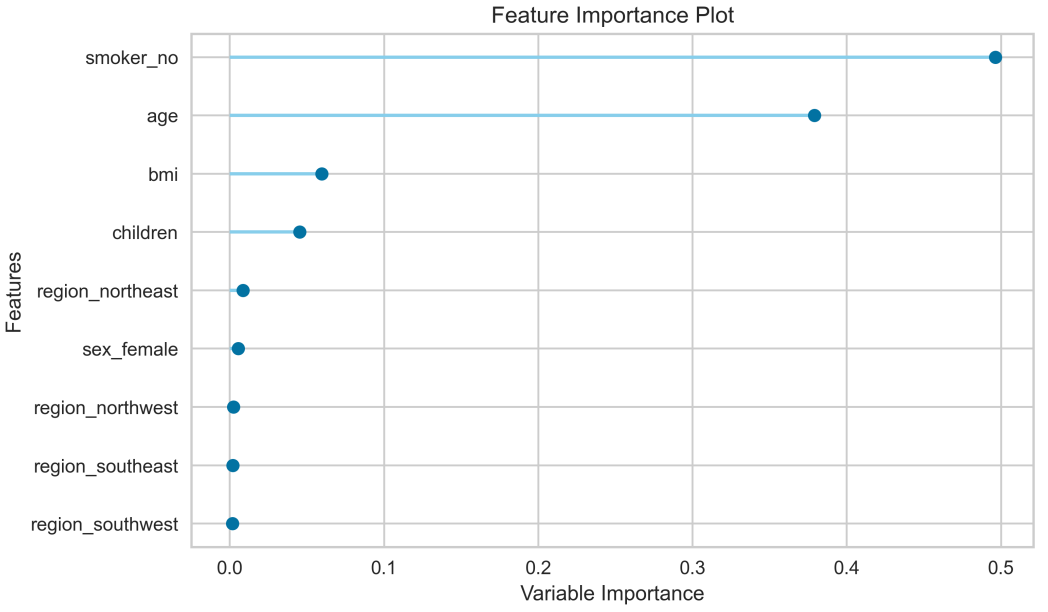
	age	bmi	children	sex_female	smoker_no	charges	Label
0	0.558673	-2.023842	0.203960	0.0	1.0	8627.541016	9174.750784
1	-0.880372	-0.739121	-1.053884	0.0	1.0	3070.808594	3816.227189
2	1.055905	-0.017651	-1.053884	0.0	1.0	10231.500000	12099.833376
3	1.197541	0.522682	-1.053884	0.0	0.0	43921.183594	43471.562081
4	0.273279	-0.619599	0.934173	1.0	0.0	22478.599609	23532.006358

So far, we have evaluated model performance by applying the k-fold cross validation technique on the train dataset. The `predict_model()` function makes predictions on the test dataset, thus letting us evaluate model performance on data that haven't been used to train the model. As we can see, the RMSE value on the test dataset is 4100.7982, indicating that the model has excellent performance.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
[4] https://scikit-learn.org/

Plotting the Model

```
plot_model(tuned_model, 'feature', scale = 4)
```



The `plot_model()` function lets us easily create various useful graphs for our machine learning model. In this case, we created a feature importance plot to visualize the impact that each feature has at predicting the target variable. Evidently, the most important features are each person's age and being a smoker or not. This confirms the EDA insights, where we concluded that those features are indeed the most important. This function can also be used to plot the validation curve, the learning curve, and various other useful graphs about the regression model.

```
plot_model(tuned_model, 'error')
```

We now use the `plot_model()` function to create a prediction error plot, visualizing the difference between the dataset target values and the predictions of our model. As we can see, the model predictions are fairly accurate, because the best fit line is significantly close to the identity line, where all predictions are theoretically perfect. Furthermore, the R^2 metric also indicates that the model is accurate, having a value of 0.892.

Finalizing and Saving the Model

```
final_model = finalize_model(tuned_model)

save_model(final_model, 'regression_model')
```

As we saw earlier, the `setup()` function splits the dataset into train and test subsets. When we create or tune a model, only the train subset is used for training, while the rest of the data are reserved for testing purposes. In case we are satisfied with the performance of our model, we can use the `finalize_model()` function to train it on the complete dataset, thus utilizing the test subset as well. After doing that, we can use the `save_model()` function to save it on the local disk. We are going to see how this stored regression model can be deployed as a web application in a following chapter.

3. Classification

Classification is one of the fundamental supervised learning tasks, where the goal is to predict a categorical variable, known as the class label. This task is known as binary classification when there are only two classes (0 and 1), or multiclass classification in case there are more. One of the most widely used binary classification models is Logistic Regression[12], which is defined in the following equation.

$$\log\left(\frac{p_n}{1-p_n}\right) = \beta_0 + \beta_1 x_{n1} + \cdots + \beta_p x_{np} = \boldsymbol{\beta}^\top \boldsymbol{X_n}$$

- $\log\left(\frac{p_n}{1-p_n}\right)$ is the natural logarithm of the odds, known as the logit function.
- x_1 to x_p are the feature variables.
- β_0 is the intercept term.
- β_1 to β_p are the coefficients of the feature variables.
- $\boldsymbol{\beta}^\top \boldsymbol{X_n}$ is the vectorized form of the equation.

Our goal is to calculate p_n, i.e. the probability that an instance of the given dataset belongs to class 1. The logistic function $\sigma(z)$ is the inverse of logit (or log-odds) function, so we can apply it and get the desired result.

$$p_n = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X_n}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^\top \boldsymbol{X_n})}$$

Classification with PyCaret

Besides Logistic Regression, there are numerous other classification models available, such as Decision Tree[13], K-Nearest Neighbors[14], Linear Discriminant Analysis[15] and XGBoost[16]. In this chapter, we are going to see how the PyCaret library can help us determine which model is optimal for our dataset. We are also going to apply Exploratory Data Analysis (EDA), so we get a better understanding of the data by visualizing their statistical properties.

Importing the Necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from pycaret.datasets import get_data
from pycaret.classification import *
mpl.rcParams['figure.dpi'] = 300
```

We begin by importing the necessary Python libraries. Pandas, Matplotlib and Seaborn are some standard libraries used in machine learning and data science. Furthermore, we import all the functions of the PyCaret classification module.

Loading the Dataset

As we first saw in the Regression chapter, PyCaret includes numerous datasets that are suitable for machine learning tasks. In this project, we are going to use Iris[1], one of the most popular datasets used to train and test classification models, freely provided by the UCI Machine Learning Repository[2]. The dataset classes are Iris Setosa, Iris Versicolor and Iris Virginica, which are all species of the plant genus Iris. The features include sepal length, sepal width, petal length and petal width of each plant. As we mentioned before, real-world data are typically more complicated, but working with toy datasets like Iris can help beginners become familiar with machine learning, and get a grasp of the associated techniques.

[1] https://archive.ics.uci.edu/ml/datasets/Iris
[2] https://archive.ics.uci.edu/ml/index.php

```
data = get_data('iris')
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

We use the `get_data()` PyCaret function to load the Iris dataset to a pandas dataframe. The first 5 rows are then printed, which is equivalent to the output of the pandas `head()` function.

```
data.info()
```

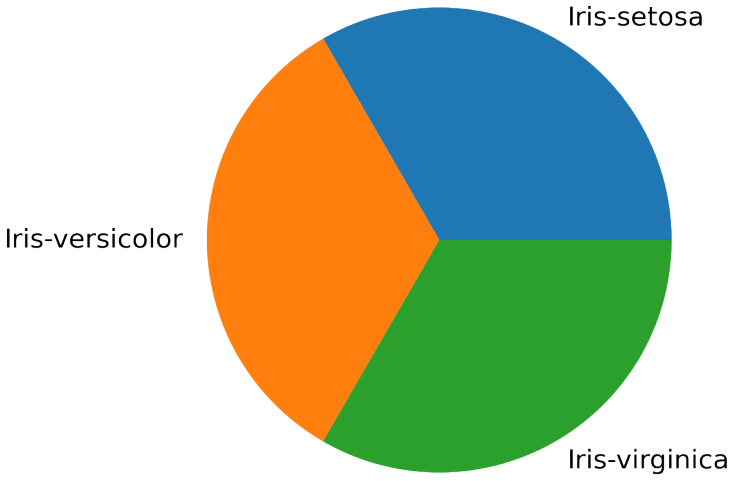
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

The `info()` pandas function lets us examine some basic information about the dataset. we can see that there are 150 rows and 5 columns. All the feature variables are numeric, while the target variable is categorical as expected. The data types have been inferred automatically, but they can be changed if needed.

Exploratory Data Analysis

We are now going to perform Exploratory Data Analysis (EDA) on the Iris dataset. EDA is a fundamental part of every machine learning project, as it helps us understand the fundamental statistical properties of a dataset, by using visualizations.

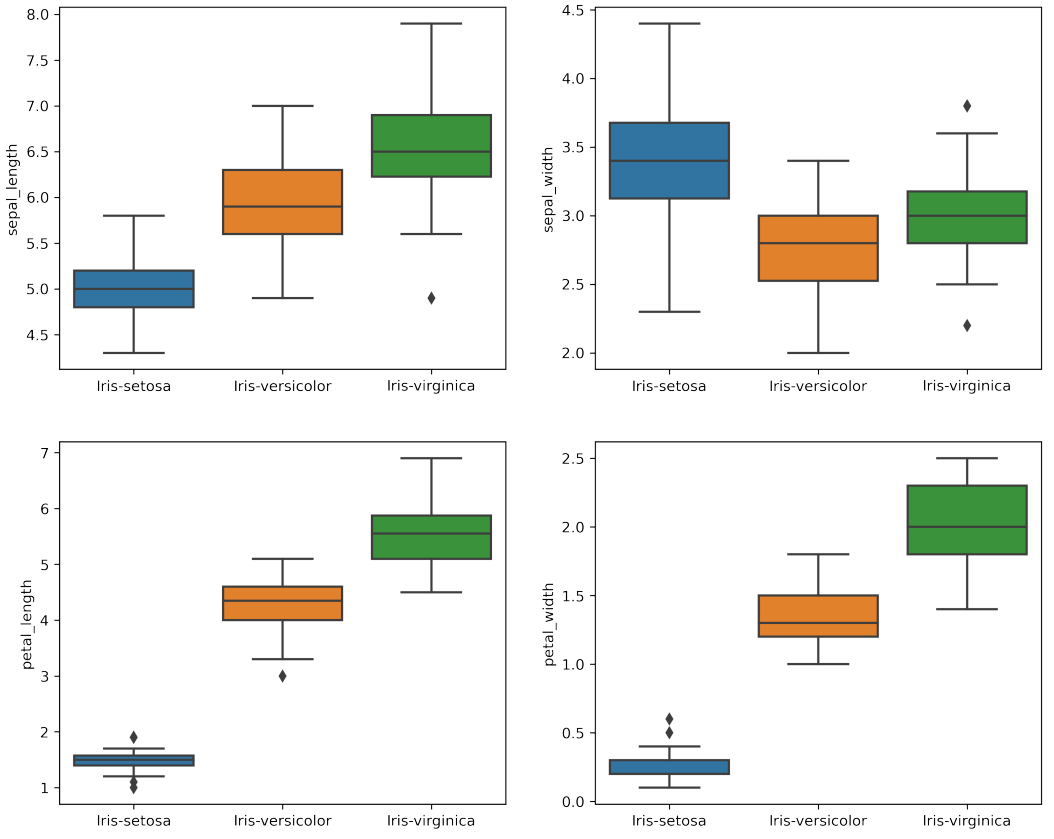
```
data['species'].value_counts().plot(kind='pie')
plt.ylabel('')
plt.show()
```



Pie charts let us easily visualize the proportions of categorical variables. As we can see, the Iris classes are evenly distributed, each one being 33.3% of the dataset. This is beneficial, as most classification models perform optimally with balanced classes. In case you are working with imbalanced data, you can refer to the PyCaret documentation[3], as it supports various ways to fix them, such as the SMOTE technique.

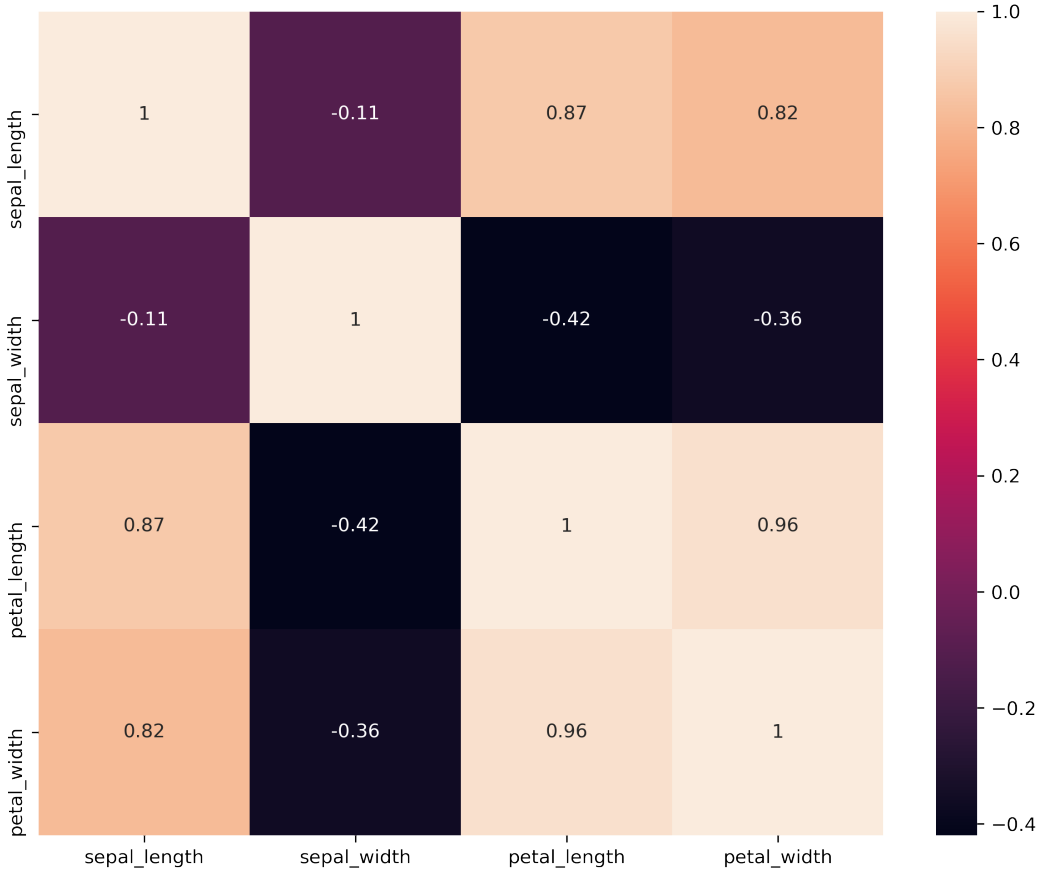
[3] https://pycaret.org/fix-imbalance/

```
fig, axes = plt.subplots(2, 2, figsize = (12, 10))

for ax, col in zip(axes.flatten(), data.columns) :
    sns.boxplot(data = data, x = 'species', y = col, ax = ax)
    ax.set_xlabel('')
```



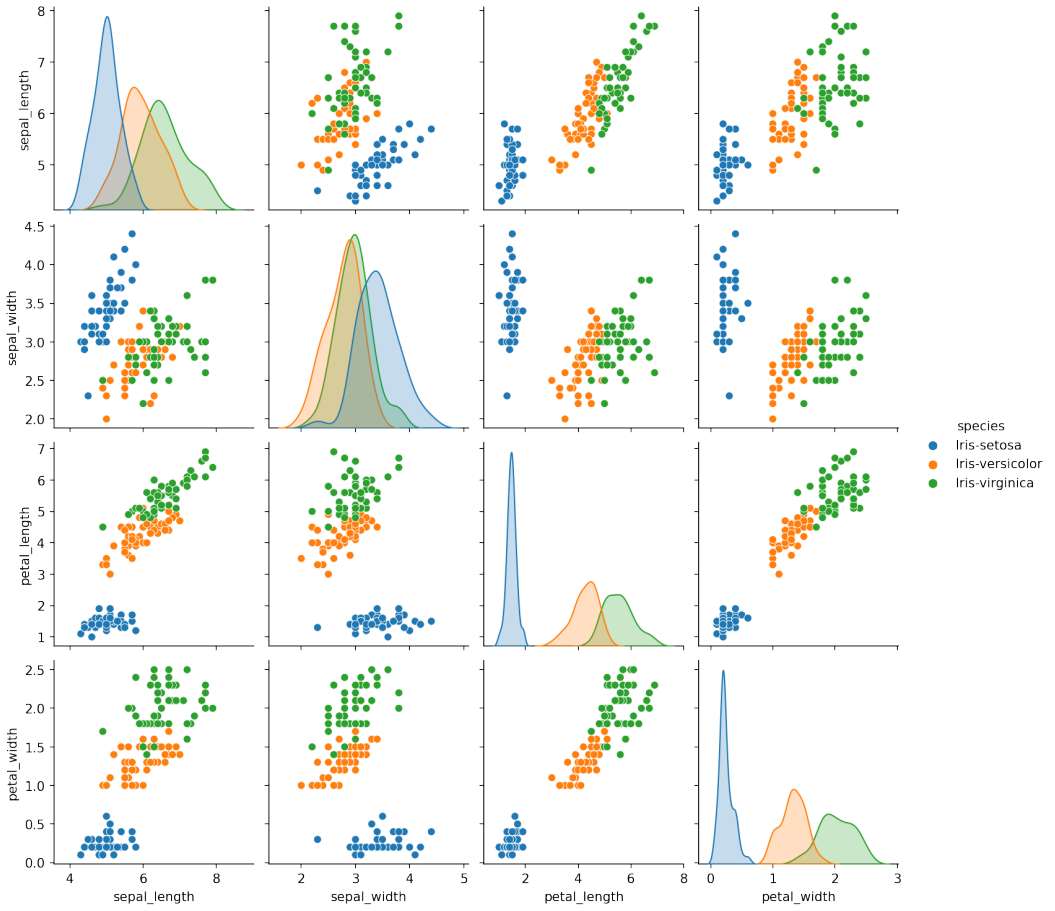
The box plot is a type of graph that visualizes the distribution of numeric variables, in a way that enables comparisons between categories. Box plots show the quartiles of each variable, including the min, Q1, median, Q3 and max values. In this case, we used the Seaborn `boxplot()` function to create box plots for all feature variables, with each plot representing a class of the target variable. Evidently, each class has a significantly different distribution, especially for the petal variables. Furthermore, there are some outliers present, visualized as diamond symbols.

```
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr().round(decimals=2), annot=True)
plt.show()
```



The Pearson correlation coefficient is a measure of correlation between two numeric variables. It has a value between -1 and 1, with 1 indicating a perfect linear relationship, and -1 indicating an inverse linear relationship. The pandas `corr()` function returns the correlation values between all numeric variables of a dataframe. Furthermore, we can use the Seaborn `heatmap()` function to plot the correlation values as a color-encoded matrix. Evidently, some of the variables have strong linear relationships, e.g. petal width and petal length. On the other hand, sepal width and petal length have an inverse linear relationship, with a correlation value of -0.42.

```
sns.pairplot(data, hue='species')
plt.show()
```



Scatter plot graphs visualize the relationship between numeric variables, helping us understand if they are correlated. Furthermore, a scatter plot matrix is a grid of scatter plots for all pairs of numeric variables in a dataset. We used the Seaborn `pairplot()` function to create a scatter plot matrix for the Iris dataset, with the diagonal being a KDE plot of each variable. We also applied hue mapping to highlight the differences between each class. Evidently, some of the variables have strong linear relationships, as indicated earlier by the Pearson correlation coefficient. Furthermore, we can see that Iris setosa is clearly separated from the other two classes.

Initializing the PyCaret Environment

```
classf = setup(data = data, target = 'species', train_size = 0.8,
normalize = True, session_id = 3934)
```

	Description	Value
0	session_id	3934
1	Target	species
2	Target Type	Multiclass
3	Label Encoded	Iris-setosa: 0, Iris-versicolor: 1, Iris-virginica: 2
4	Original Data	(150, 5)
5	Missing Values	False
6	Numeric Features	4
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(120, 4)
12	Transformed Test Set	(30, 4)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False

After completing the Exploratory Data Analysis, we need to initialize the PyCaret environment. This can be accomplished easily with the `setup()` function, which prepares the data for model training. This function has numerous parameters, enabling you to create a complete data preprocessing pipeline. Regardless of that, we are only going to examine the most important functionality to keep things simple. Please check the PyCaret Classification module[4] documentation page for more details. After running `setup()`, we get a table of useful information about its settings and parameters, explained in detail below.

Identifying Numeric and Categorical Features

PyCaret can automatically infer whether a feature is numeric or categorical. When you execute the `setup()` function, you will be prompted to verify that the features have been identified correctly. Alternatively, you can specify which features are categorical or numeric using the `categorical_features` and `numeric_features` parameters. In this case, all the features have been correctly identified as numeric, so there is no need for any changes.

Train/Test Split

Splitting a dataset to a train and test subset is standard practice in machine learning. We have set the `train_size` parameter to 0.8, meaning that the machine learning model will be trained on 80% of the original data, while the 20% will be used for testing purposes.

Normalization of Numeric Features

Some classification models require numeric features to be normalized, so they can perform optimally. Normalized features have $\mu = 0$ and $\sigma = 1$. We can accomplish that by replacing each value of the feature with the z-score, which is defined as $z = \frac{x-\mu}{\sigma}$. We enabled normalization by setting the `normalize` parameter to `True`.

[4] https://pycaret.readthedocs.io/en/latest/api/classification.html

Comparing Classification Models

```
compare_models(sort = 'Accuracy')
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lda	Linear Discriminant Analysis	0.9833	1.0000	0.9833	0.9867	0.9831	0.9750	0.9769	0.0050
qda	Quadratic Discriminant Analysis	0.9750	1.0000	0.9750	0.9800	0.9746	0.9625	0.9653	0.0040
lr	Logistic Regression	0.9583	1.0000	0.9617	0.9697	0.9574	0.9375	0.9436	0.0060
knn	K Neighbors Classifier	0.9583	0.9932	0.9600	0.9671	0.9579	0.9375	0.9421	0.0270
nb	Naive Bayes	0.9583	0.9979	0.9617	0.9697	0.9574	0.9375	0.9436	0.0050
et	Extra Trees Classifier	0.9583	1.0000	0.9600	0.9693	0.9571	0.9375	0.9436	0.0900
lightgbm	Light Gradient Boosting Machine	0.9500	0.9917	0.9517	0.9626	0.9486	0.9250	0.9320	0.0330
xgboost	Extreme Gradient Boosting	0.9417	0.9855	0.9467	0.9564	0.9406	0.9124	0.9203	0.0630
catboost	CatBoost Classifier	0.9417	0.9990	0.9467	0.9564	0.9406	0.9124	0.9203	0.4250
rf	Random Forest Classifier	0.9333	0.9938	0.9400	0.9526	0.9321	0.9004	0.9106	0.1010
gbc	Gradient Boosting Classifier	0.9333	0.9896	0.9383	0.9526	0.9320	0.9005	0.9108	0.0300
dt	Decision Tree Classifier	0.9250	0.9442	0.9256	0.9393	0.9235	0.8870	0.8949	0.0050
ada	Ada Boost Classifier	0.9167	0.9927	0.9233	0.9415	0.9143	0.8754	0.8890	0.0160

The `compare_models()` function simplifies the process of choosing the best classification model, by training all available models and printing the results. The first column is the model name and the rest are various performance metrics. The models are sorted based on the metric of our preference, which is accuracy in this case. Simply put, accuracy is the percentage of correct predictions that the classification model managed to make. It is defined in the following formula for binary classification, but it extends to multiple classes as well. Accuracy is a useful metric only for balanced classes, so you'll have to choose a different one for imbalanced datasets. We can see that the Linear Discriminant Analysis model performed the best, with an accuracy of 98.33%.

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive\ + True\ Negative + False\ Positive + False\ Negative}$$

Creating the Model

```
model = create_model('lda')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9167	1.0000	0.9167	0.9333	0.9153	0.8750	0.8843
3	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	0.9167	1.0000	0.9167	0.9333	0.9153	0.8750	0.8843
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	0.9833	1.0000	0.9833	0.9867	0.9831	0.9750	0.9769
SD	0.0333	0.0000	0.0333	0.0267	0.0339	0.0500	0.0463

We are now going to use the `create_model()` function to train the Linear Discriminant Analysis model, as it performed best in the model comparison. This function uses stratified k-fold cross validation to evaluate model accuracy, a variation of the standard k-fold technique that was used in the Regression chapter. The dataset is consecutively partitioned into k subsamples, with one subsample being retained for validation, while the rest are used to train the model. The difference between stratified k-fold and standard k-fold is that subsamples are stratified to preserve the percentage of samples for each class, in this case 33.3% for every Iris species. After the stratified k-fold cross validation is completed, the resulting metrics for each subsample are printed, including the mean and standard deviation values.

Tuning the Model

```
model_cat = create_model('catboost', verbose = False)

params = {'iterations': np.arange(100, 1000, 100),
          'max_depth': np.arange(1, 10),
          'learning_rate': np.arange(0.01, 1, 0.01),
          'random_strength': np.arange(0.1, 1.0, 0.1),
          'l2_leaf_reg': np.arange(1, 100),
          'border_count': np.arange(1, 256)}

tuned_model = tune_model(model_cat, optimize = 'Accuracy', fold = 10,
              tuner_verbose = False, search_library = 'scikit-optimize',
              custom_grid = params, n_iter = 50)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9167	1.0000	0.9167	0.9333	0.9153	0.8750	0.8843
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9167	0.9896	0.9167	0.9333	0.9153	0.8750	0.8843
3	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	0.8333	1.0000	0.8333	0.8889	0.8222	0.7500	0.7833
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	0.9167	1.0000	0.9333	0.9333	0.9167	0.8737	0.8830
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	0.9167	1.0000	0.9333	0.9375	0.9180	0.8750	0.8843
Mean	0.9500	0.9990	0.9533	0.9626	0.9488	0.9249	0.9319
SD	0.0553	0.0031	0.0536	0.0395	0.0577	0.0830	0.0738

The Linear Discriminant Analysis model has excellent performance, with an accuracy of 98.33%. Unfortunately, not every model performed as well, with the CatBoost Classifier model achieving an accuracy of 94.17%. The `tune_model()` function can improve the performance of a classification model by finding the optimal combination of its hyperparameters, a technique known as hyperparameter tuning. By default, this function uses the randomized search method[5] of the scikit-learn library, but there are various other alternatives, including the scikit-optimize[6] and optuna[7] libraries. In this case, we selected the scikit-optimize library that uses Bayesian Optimization[17] to model the hyperparameter search space, thus finding the optimal combination as soon as possible.

First of all, we trained a CatBoost classifier model on the Iris dataset. After doing that, we passed the model to the `tune_model()` function, along with a dictionary including the hyperparameters of our preference. Furthermore, we used the NumPy `arange()` function to set the range of values for each hyperparameter. We can see that after tuning the model, the accuracy increased to 95%. This isn't a huge improvement, so we could try experimenting with different hyperparameters or range of values for each one of them.

It should also be noted that hyperparameter tuning is a computationally intensive and time-consuming process, as the model has to be trained numerous times to get the optimal result. In case it takes too long to complete on your computer, try decreasing the number of iterations by setting a lower value on the `n_iter` parameter. Finally, each classification model has a different set of hyperparameters that can be tuned, so you'll have to consult its documentation to select them. In this case, you can refer to the CatBoost Parameter tuning[8] documentation page.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
[6] https://scikit-optimize.github.io/stable/index.html
[7] https://optuna.org/
[8] https://catboost.ai/docs/concepts/parameter-tuning.html

Making Predictions

```
predictions = predict_model(model)
```

```
predictions.head(10)
```

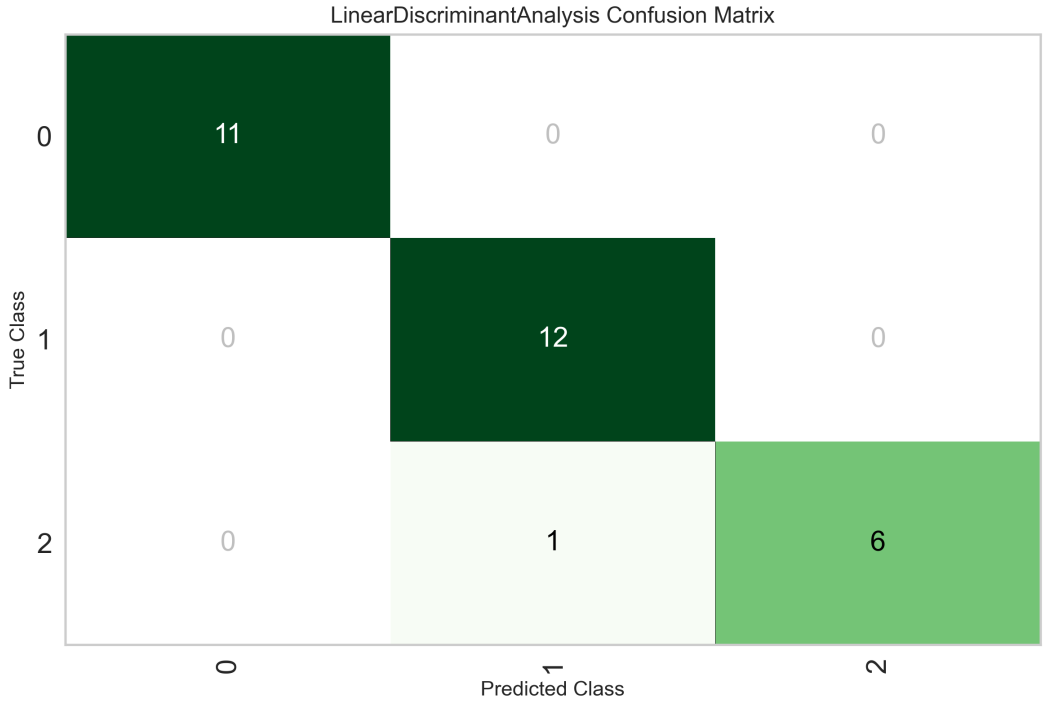
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Linear Discriminant Analysis	0.9667	1.0000	0.9524	0.9692	0.9661	0.9484	0.9501

	sepal_length	sepal_width	petal_length	petal_width	species	Label	Score
0	-0.085637	2.256249	-1.480436	-1.354654	Iris-setosa	Iris-setosa	1.000
1	1.198925	0.114993	0.883009	1.120194	Iris-virginica	Iris-virginica	0.999
2	-0.319194	-1.312511	0.038922	-0.182357	Iris-versicolor	Iris-versicolor	1.000
3	0.615033	0.352911	0.826737	1.380705	Iris-virginica	Iris-virginica	1.000
4	-0.435973	2.732084	-1.367891	-1.354654	Iris-setosa	Iris-setosa	1.000
5	-1.136643	-0.122924	-1.367891	-1.354654	Iris-setosa	Iris-setosa	1.000
6	-0.786308	1.066663	-1.311618	-1.354654	Iris-setosa	Iris-setosa	1.000
7	-0.903086	1.780415	-1.255346	-1.354654	Iris-setosa	Iris-setosa	1.000
8	-0.085637	-1.074594	0.095194	-0.052102	Iris-versicolor	Iris-versicolor	1.000
9	-0.903086	1.542498	-1.311618	-1.094143	Iris-setosa	Iris-setosa	1.000

Up to this point, we have used stratified k-fold cross validation to evaluate model accuracy on the train dataset. The `predict_model()` function makes predictions on the test dataset that was created during the PyCaret environment initialization, thus letting us evaluate model accuracy on data that haven't been used to train the model. As we can see, the Linear Discriminant Analysis model has an accuracy of 96.67% on the test dataset, which is very close to the cross validation results. Furthermore, the function returns a dataframe with the dataset columns, as well as the predicted class for each instance.

Plotting the Model

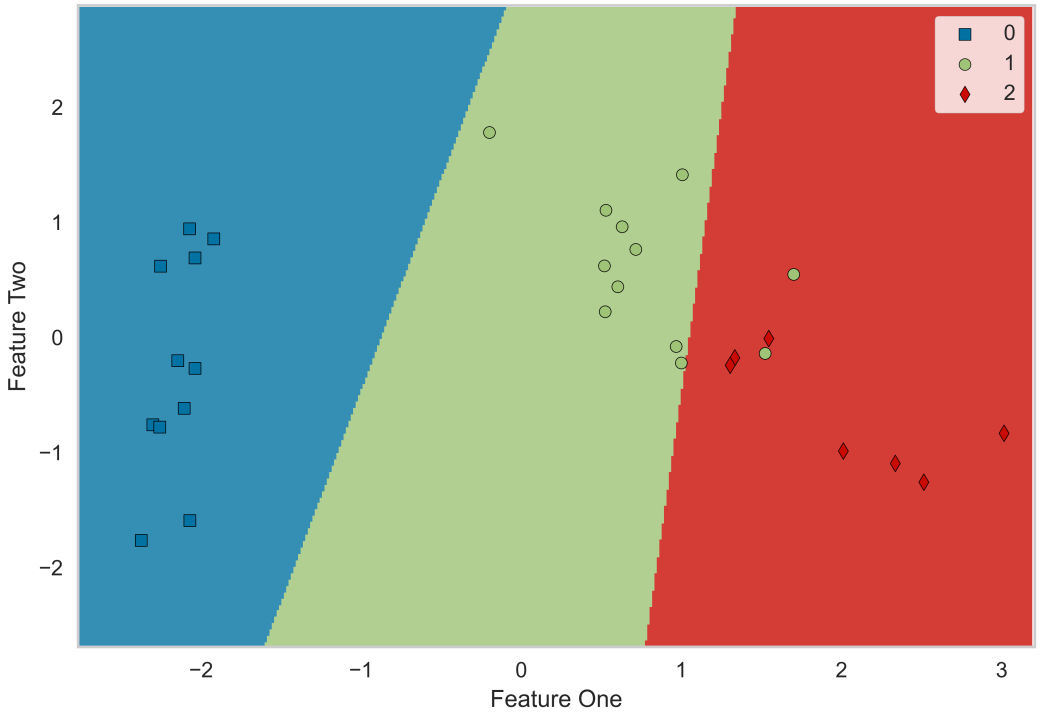
```
plot_model(model, 'confusion_matrix')
```



The `plot_model()` function lets us create various useful graphs for our classification model. In this case, we created a confusion matrix that visualizes the accuracy of the Linear Discriminant Analysis model. Each column of the matrix represents the predicted class, while each row is the actual class of the test set instances. We can see that only one instance was predicted incorrectly, indicating the high accuracy of the LDA model. For a complete list of the available plots, please refer to the Classification[9] page of the PyCaret documentation.

[9] https://pycaret.readthedocs.io/en/latest/api/classification.html#pycaret.classification.plot_model

```
plot_model(model, 'boundary', scale = 2)
```



We now use the `plot_model()` function to plot the decision boundaries that partition the vector space into different sets, one for each class. As we can see, one of those classes is clearly separated from the rest, while the others are not so easily distinguishable. This was previously observed on the scatter plot matrix that we created in the EDA section.

Finalizing and Saving the Model

```
final_model = finalize_model(model)

save_model(final_model, 'classification_model')
```

As we saw earlier, the `setup()` function splits the dataset into train and test subsets. When we create or tune a model, only the train subset is used for training, while the rest of the data are reserved for testing purposes. In case we are satisfied with the performance of our model, we can use the `finalize_model()` function to train it on the complete dataset, thus utilizing the test subset as well. After doing that, we can use the `save_model()` function to save it on the local disk. We are going to see how stored models can be deployed as a web application in a following chapter.

4. Clustering

One of the fundamental tasks in unsupervised machine learning is clustering. The goal in this task is to categorize instances of a given dataset in different clusters, based on their common characteristics. Clustering has many practical applications in various fields, including market research, social network analysis, bioinformatics, medicine and others. K-Means clustering[18] is a simple and widely used method, as defined in the following formula.

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W\left(C_k\right) \right\}$$

K is the number of all clusters, while C_K represents each individual cluster. Our goal is to minimize W, which is the measure of within-cluster variation.

$$W\left(C_k\right) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2$$

There are various ways to define within-cluster variation, but the most common one is squared euclidean distance, as seen in the above equation. This results in the following form of K-Means clustering.

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 \right\}$$

Clustering with PyCaret

K-Means is a widely used method, but there are numerous others available, such as Affinity Propagation[19], Spectral Clustering[20], Agglomerative Clustering[21], Mean Shift Clustering[22] and Density-Based Spatial Clustering (DBSCAN)[23]. In this chapter, we are going to see how the PyCaret clustering module can help us easily train a model and evaluate its performance.

Importing the Necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from pycaret.clustering import *
from sklearn.datasets import make_blobs
mpl.rcParams['figure.dpi'] = 300
```

As in previous chapters, we begin by importing some standard Python libraries, including NumPy, pandas, Matplotlib and Seaborn. We also import the PyCaret clustering functions, as well as the `make_blobs()` scikit-learn function that can be used to generate datasets. Finally, we set the Matplotlib figure DPI to 300, so we get high-resolution plots for this book. Having this setting enabled isn't necessary, so you can remove the last line if you want.

Generating a Synthetic Dataset

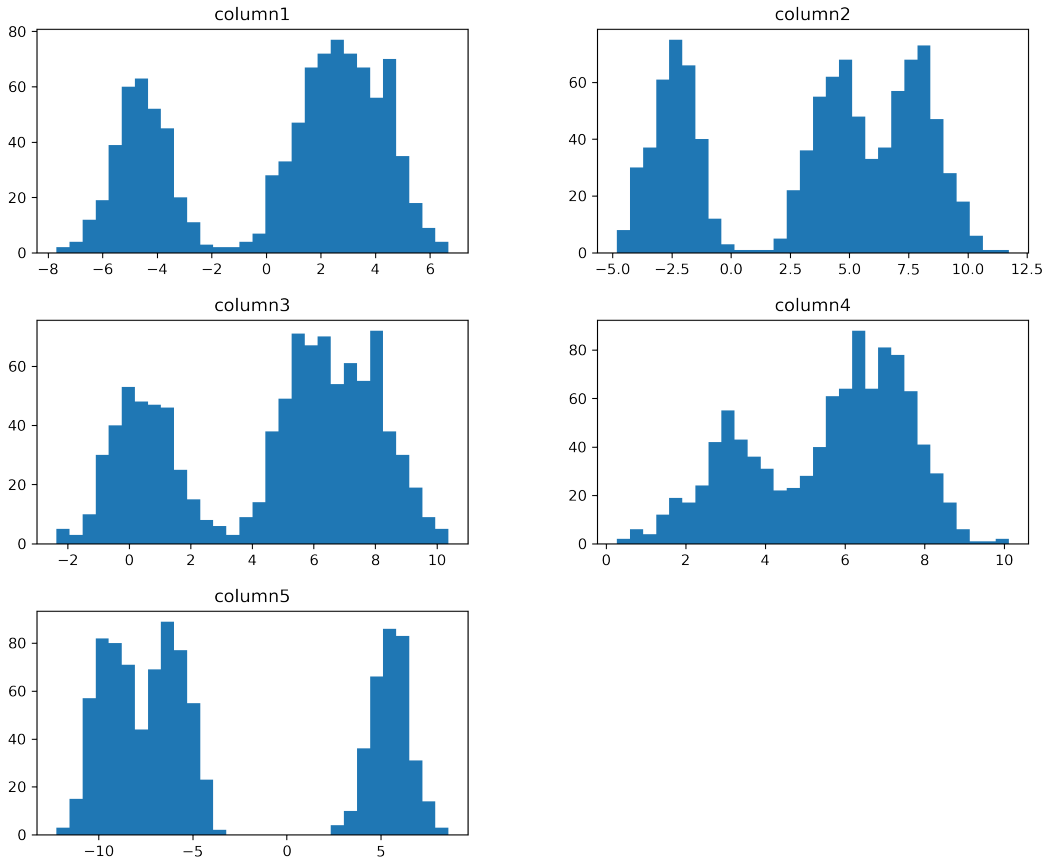
```
cols = ['column1', 'column2', 'column3',
        'column4', 'column5']

arr = make_blobs(n_samples = 1000, n_features = 5, random_state = 20,
                 centers = 3, cluster_std = 1)

data = pd.DataFrame(data = arr[0], columns = cols)
data.head()
```

	column1	column2	column3	column4	column5
0	2.916076	6.504857	8.246937	6.185952	-9.776481
1	-2.044462	5.282757	5.265467	7.487941	6.347177
2	3.439328	-3.020505	-0.783592	2.907385	-6.117881
3	2.155568	8.776136	5.788650	5.751763	-7.609132
4	-6.650020	5.458227	5.848407	7.150584	5.992247

Instead of loading a dataset, we are going to generate a synthetic one, by using the `make_blobs()` scikit-learn function. This function generates datasets that are suitable for clustering models, and has various parameters that can be modified according to our needs. In this case, we created a dataset with 1000 instances, 5 features and 3 distinct clusters. Using a synthetic dataset to test our clustering model has various benefits, the main being that we already know the actual number of clusters, so we can evaluate model performance easily. Real-world data are typically more complicated, i.e. they don't always have clearly separated clusters, but working with a simple dataset lets you become acquainted with the tools and workflow.

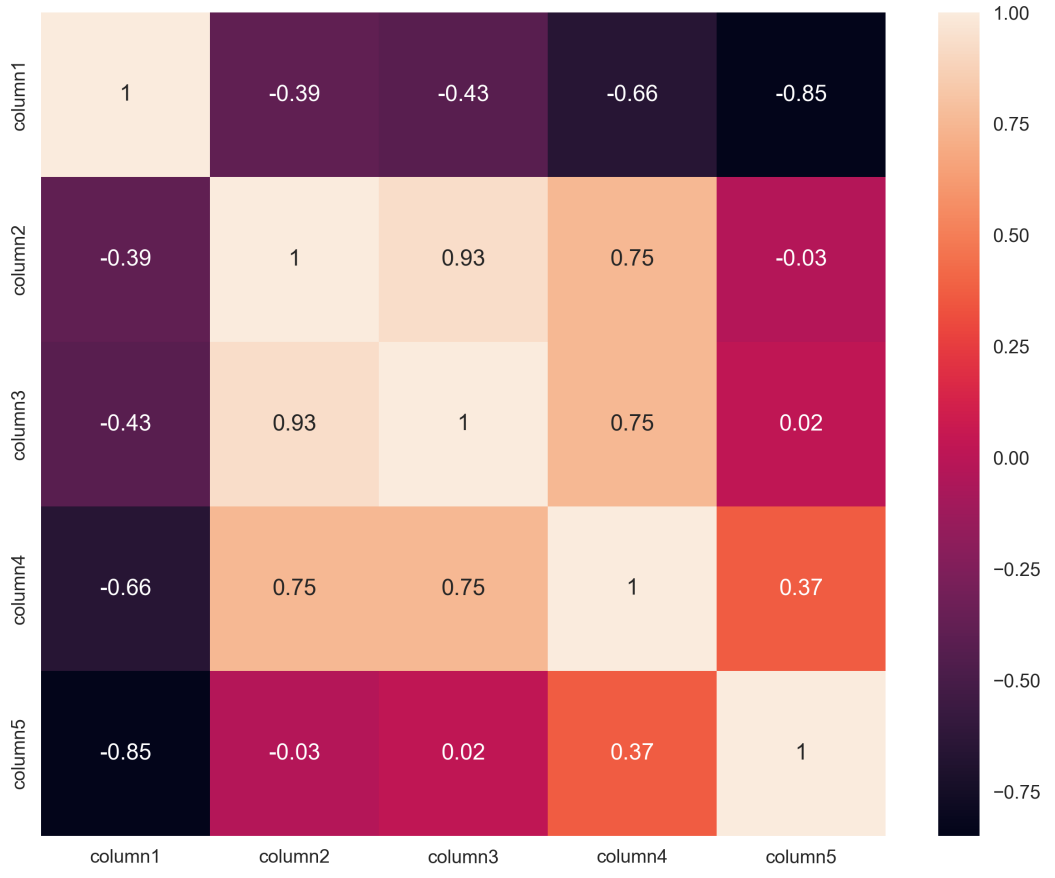
Exploratory Data Analysis

```
data.hist(bins = 30, figsize = (12,10), grid = False)

plt.show()
```



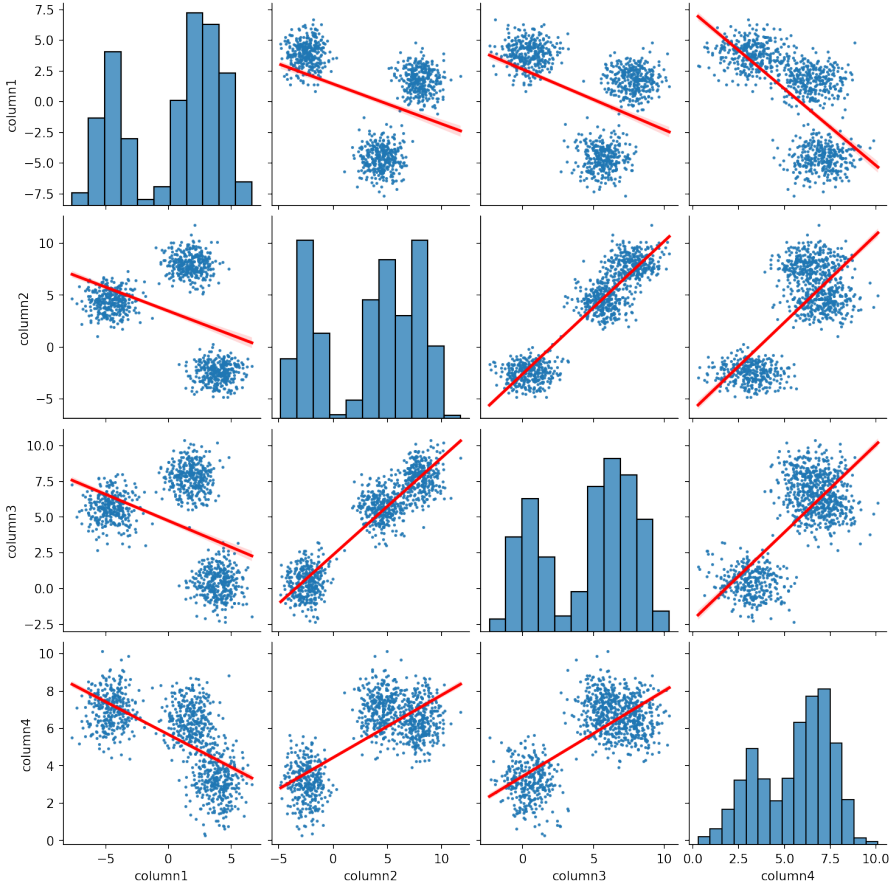
The `hist()` pandas function lets us easily visualize the distribution of each variable. We can see that all variable distributions are either bimodal or multimodal, i.e. they have two or more peaks. This typically happens when the dataset contains multiple groups with different characteristics. In this case, the dataset was specifically created to contain 3 distinct clusters, so it is reasonable for the variables to have multimodal distributions.

```
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr().round(decimals=2), annot=True)

plt.show()
```



We used the `corr()` pandas function, as well as the `heatmap()` Seaborn function to create a heatmap that visualizes the correlation values of all variable pairs. We can see that column2 and column3 have a strong linear relationship, with a correlation value of 0.93. The same is true for column3 and column4, having a correlation value of 0.75. On the other hand, column1 is inversely correlated with all the other columns, but especially column5, with a value of -0.85.

```
plot_kws = {'scatter_kws': {'s': 2}, 'line_kws': {'color': 'red'}}
sns.pairplot(data, kind='reg', vars=data.columns[:-1], plot_kws=plot_kws)
plt.show()
```



We used the `pairplot()` Seaborn function to create a scatter plot matrix for the synthetic dataset, with the diagonal having the histogram of each variable. The dataset clusters are visible in the scatter plots, indicating that they are clearly separated from each other. As it was previously observed in the correlation heatmap, we can see that some of the variable pairs have strong linear relationships, while others have inverse linear relationships. This is highlighted by the regression lines that have been included in each scatter plot, by setting the `kind` parameter of the `pairplot()` function to `reg`.

Initializing the PyCaret Environment

```
cluster = setup(data, session_id = 7652)
```

	Description	Value
0	session_id	7652
1	Original Data	(1000, 5)
2	Missing Values	False
3	Numeric Features	5
4	Categorical Features	0
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(1000, 5)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	cluster-default-name
13	USI	29a9

After completing the Exploratory Data Analysis (EDA), we are now going to use the `setup()` function to initialize the PyCaret environment. By doing this, a pipeline that prepares the data for model training and deployment will be created. In this case, the default settings are acceptable, so we aren't going to modify any of the parameters. Regardless, this powerful function has numerous data preprocessing abilities, so you can refer to the documentation page of the PyCaret Clustering module[1] to read more details about it.

[1] https://pycaret.readthedocs.io/en/latest/api/clustering.html

Creating a Model

```
model = create_model('kmeans')
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.5822	6041.4265	1.2247	0	0	0

The `create_model()` function lets us easily create and evaluate the clustering model of our preference, such as the K-Means algorithm. This function creates 4 clusters by default, so we could simply set the `num_clusters` parameter to 3, as this is the correct number. Instead of doing that, we are going to follow an approach that generalizes for real-world datasets, where the cluster number is typically unknown. After executing the function, a number of performance metrics are printed, including Silhouette[24], Calinski-Harabasz[25] and Davies-Bouldin[26]. We are going to focus on the Silhouette Coefficient, which is defined in the following equation.

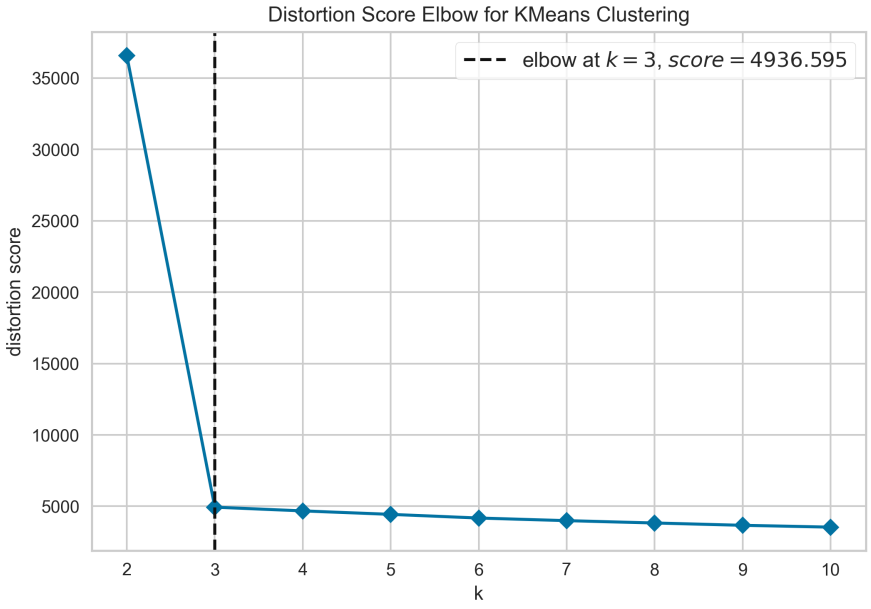
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$-1 \leq s(i) \leq 1$$

- $s(i)$ is the Silhouette Coefficient of the dataset instance i.
- $a(i)$ is the mean intra-cluster distance of i.
- $b(i)$ is the mean nearest-cluster distance of i.

The resulting metric value is the mean Silhouette Coefficient of all instances, having a range between -1 and 1. Negative values indicate that an instance has been assigned to the wrong cluster, while values near 0 indicate that clusters are overlapping. On the other hand, positive values close to 1 indicate correct assignment. In our example, the value is 0.5822, suggesting that model performance can be improved by finding the optimal number of clusters for the dataset. Next, we are going to see how we can accomplish that by using the elbow method.

```
plot_model(model, 'elbow')
```



The `plot_model()` function lets us create various useful graphs for our model. In this case, we created an elbow plot that will help us find the optimal number of clusters for the K-Means model. The elbow method trains the clustering model for a range of K values, and visualizes the distortion score for each one of them[27]. The point of inflection on the curve –known as the elbow– is an indication of the optimal value for K. As expected, the plot has an elbow at $K = 3$, highlighted by the dashed vertical line.

```
model = create_model('kmeans', num_clusters = 3)
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.7972	8565.5114	0.2904	0	0	0

After using the elbow method to find the optimal number of clusters, we train the K-Means model again. As we can see, the mean Silhouette Coefficient increased to 0.7972, indicating improved model performance and better cluster assignment for each dataset instance.

Plotting the Model

```
plot_model(model, 'cluster')
```

As seen earlier, `plot_model()` is a useful function that can be used to plot various kinds of graphs for our clustering model. In this case, we created a 2D Principal Component Analysis (PCA) plot for the K-Means model. PCA can be used to project data to a lower-dimensional space while preserving most of the variance[28], a technique known as dimensionality reduction. After applying PCA to the synthetic dataset, the original 5 features have been reduced to 2 principal components. Furthermore, we can see that the clusters are clearly separated, and all of the dataset instances have been assigned to the correct cluster.

Saving and Assigning the Model

```
save_model(model, 'clustering_model')

results = assign_model(model)
results.head(10)
```

	column1	column2	column3	column4	column5	Cluster
0	2.916076	6.504857	8.246937	6.185952	-9.776481	Cluster 2
1	-2.044462	5.282757	5.265467	7.487941	6.347177	Cluster 1
2	3.439328	-3.020505	-0.783592	2.907385	-6.117881	Cluster 0
3	2.155568	8.776136	5.788650	5.751763	-7.609132	Cluster 2
4	-6.650020	5.458227	5.848407	7.150584	5.992247	Cluster 1
5	2.845247	8.015184	9.271296	6.060792	-8.281544	Cluster 2
6	-6.252012	3.910582	4.499606	6.197325	5.454781	Cluster 1
7	-4.962223	2.730723	6.787934	7.299566	5.073320	Cluster 1
8	1.727624	6.452070	6.378151	4.688654	-9.605882	Cluster 2
9	-5.568622	4.162203	7.741415	7.373318	5.787458	Cluster 1

The `save_model()` function lets us save the clustering model to the local disk for future use or deployment as an application. The model is stored as a pickle file that can be loaded using the complementary `load_model()` function. Furthermore, the `assign_model()` function returns the synthetic dataset with an additional column for the cluster labels that were assigned to the dataset instances.

5. Anomaly Detection

Anomaly detection is one of the main tasks in unsupervised machine learning, where the goal is to identify dataset instances that differ significantly from the majority. Those instances are known as outliers, and there are various incentives to detect them, depending on the context and domain of each application. There are also semi-supervised and fully supervised methods for anomaly detection, but we are going to focus on the unsupervised approach, as it is supported by PyCaret. Local Outlier Factor[29] is one of the main anomaly detection models, as defined in the following equation.

$$\mathrm{LOF}_k(A) = \frac{\sum_{B \in N_k(A)} \frac{\mathrm{lrd}_k(B)}{\mathrm{lrd}_k(A)}}{|\mathrm{N}_k(A)|}$$

- $\mathrm{LOF}_k(\mathrm{A})$ is the Local Outlier Factor of the dataset instance A.
- $\mathrm{lrd}_k(\mathrm{A})$ is the local reachability density of A.
- k is the number of nearest neighbors to A.
- $\mathrm{N}_k(\mathrm{A})$ is the set of k nearest neighbors.

The Local Outlier Factor of an instance is the average local reachability density of its neighbors, divided by the local reachability of the instance itself. Values that are significantly larger than 1 indicate that the instance is an outlier, while smaller values suggest that it is an inlier.

Anomaly Detection with PyCaret

As mentioned earlier, Local Outlier Factor is a popular anomaly detection model, but there are numerous others available, including Isolation Forest[30], K-Nearest Neighbors Detector[31], Subspace Outlier Detection[32] and Clustering-Based Local Outlier[33]. In the rest of thic chapter, we are going to see how you can easily train and plot an anomaly detection model using the PyCaret library.

Importing the Necessary Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from pycaret.datasets import get_data
from pycaret.anomaly import *
mpl.rcParams['figure.dpi'] = 300
```

As in previous chapters, we begin by importing the libraries that are necessary for this project, including pandas, Matplotlib, Seaborn and the PyCaret Anomaly Detection module. We also imported the `get_data()` function that lets us load the dataset of our preference. Finally, we set the Matplotlib figure DPI to 300, so we can get high-quality images for this book.

Loading the Dataset

In this chapter, we are going to use the Wholesale Customers dataset[1], provided freely by the UCI Machine Learning Repository[2]. This dataset includes the annual spending of a wholesale distributor's customers, on a number of product categories. The dataset features include Fresh, Milk, Grocery, Frozen, Detergents & Paper and Delicatessen, which constitute the product categories and are all numeric variables. Furthermore, Channel and Region are categorical variables, referring to the attributes of each customer. The Channel categories include Horeca (Hotel/Restaurant/Cafe) and Retail, while the Region categories are Lisbon, Oporto and Other.

[1] https://archive.ics.uci.edu/ml/datasets/wholesale+customers
[2] https://archive.ics.uci.edu/ml/index.php

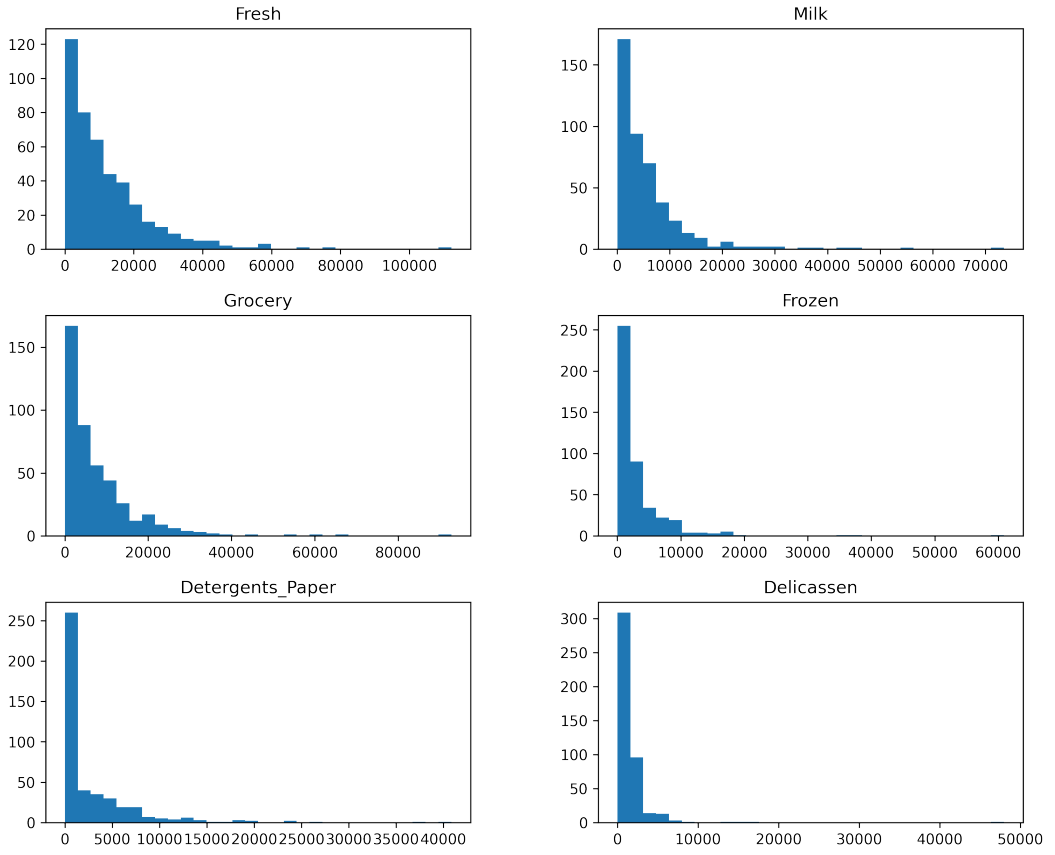
```
numeric = ['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicassen']
categorical = ['Channel','Region']

replace_dict = { "Channel": {1: "Horeca", 2: "Retail"},
                 "Region":  {1: "Lisbon", 2: "Oporto", 3: "Other"} }

data = get_data('wholesale', verbose = False)
data.replace(replace_dict, inplace = True)
data.head(10)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	Retail	Other	12669	9656	7561	214	2674	1338
1	Retail	Other	7057	9810	9568	1762	3293	1776
2	Retail	Other	6353	8808	7684	2405	3516	7844
3	Horeca	Other	13265	1196	4221	6404	507	1788
4	Retail	Other	22615	5410	7198	3915	1777	5185
5	Retail	Other	9413	8259	5126	666	1795	1451
6	Retail	Other	12126	3199	6975	480	3140	545
7	Retail	Other	7579	4956	9426	1669	3321	2566
8	Horeca	Other	5963	3648	6192	425	1716	750
9	Retail	Other	6006	11093	18881	1159	7425	2098

We use the `get_data()` function to load the Wholesale Customers dataset to a pandas dataframe. The categorical variables have been encoded as integer numbers, so we are going to restore the original categories with the `replace()` function. This may seem redundant, as the categorical variables will be encoded again during the PyCaret environment initialization, so the anomaly detection model can be trained. Regardless, having the original categories available during the Exploratory Data Analysis section of the project, will be more informative and helpful for us. After executing the `replace()` function, we use the `head()` pandas function to display the first rows of the dataset.

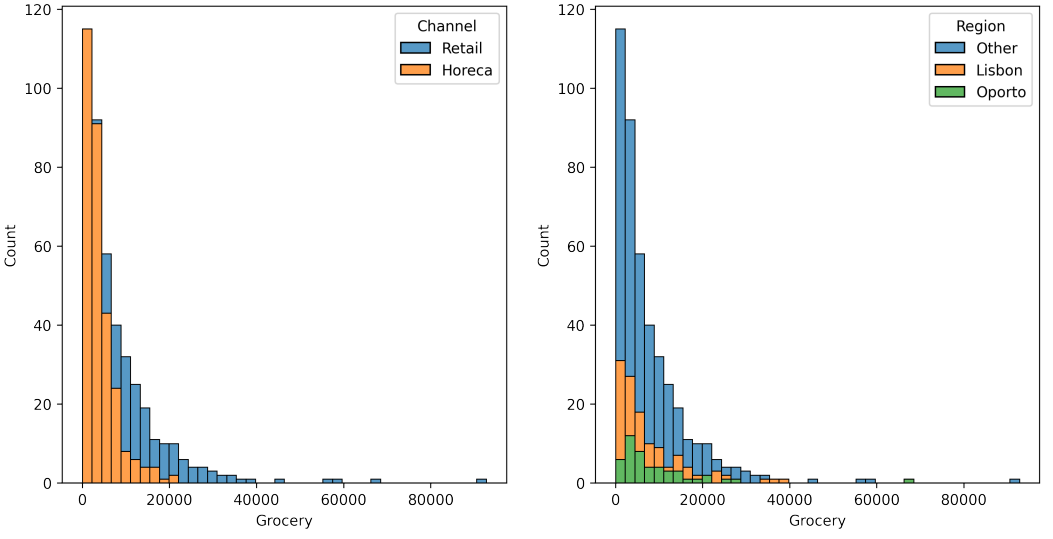
Exploratory Data Analysis

```
data[numeric].hist(bins=30, figsize=(12,10), grid=False)

plt.show()
```



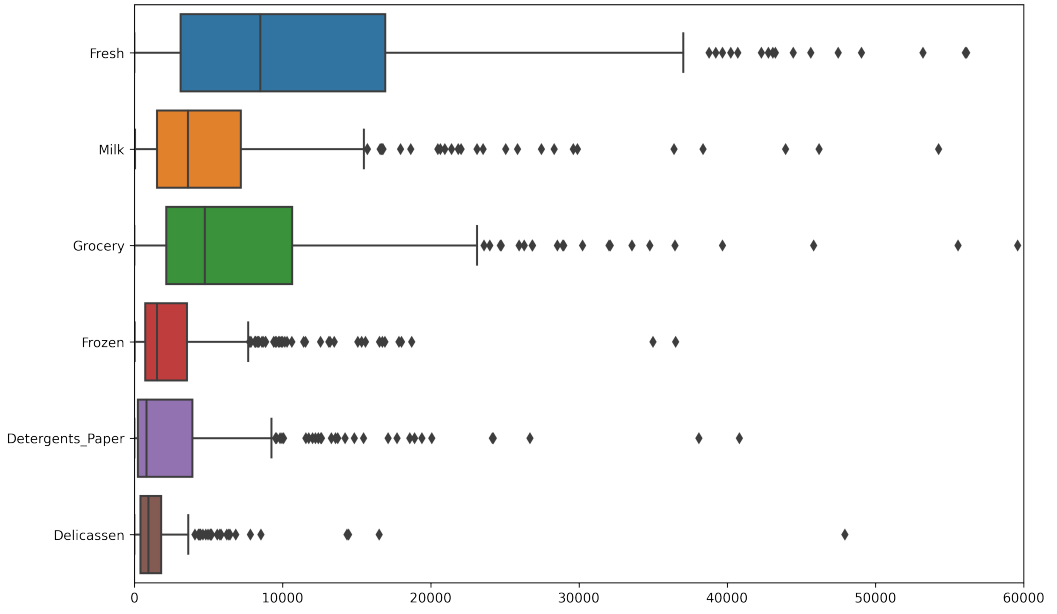
We begin the EDA section of the project by visualizing the distribution of numeric features, a task that can be easily accomplished with the `hist()` pandas function. Evidently, all the features are extremely right-skewed, indicating that some instances have significantly higher values than the rest, thus being outliers.

```
fig, axes = plt.subplots(1, 2, figsize = (12,6))

for ax, col in zip(axes.flatten(), categorical):
    sns.histplot(data=data, x ='Grocery', hue=col,
                 multiple='stack', ax=ax)

plt.show()
```



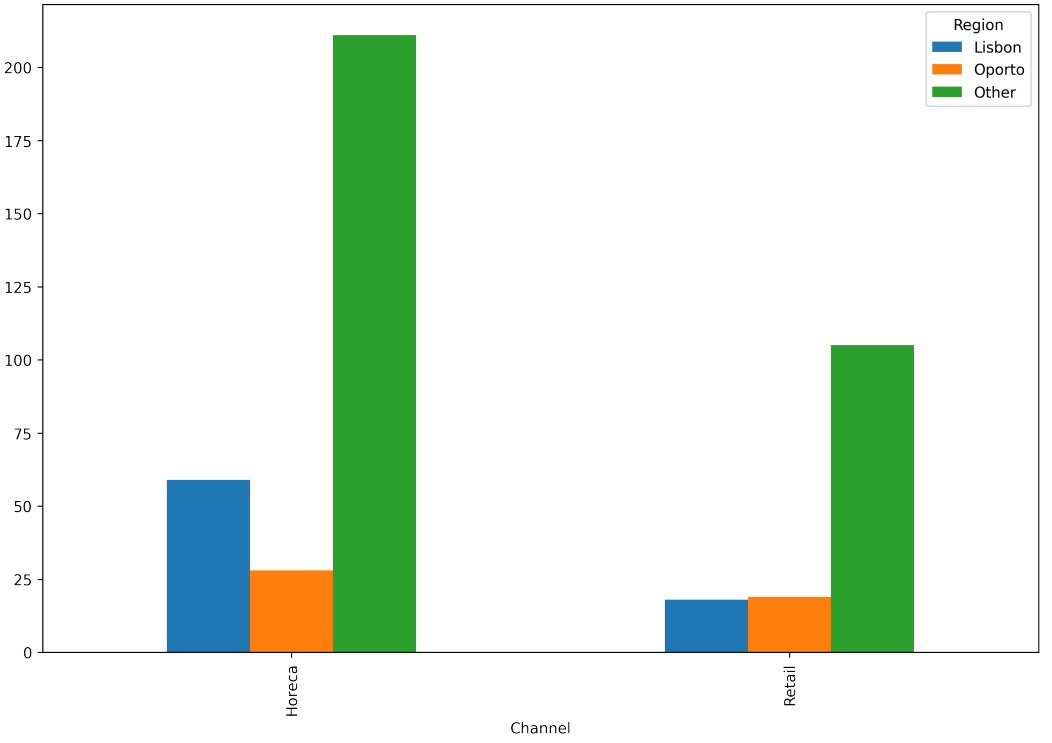
Visualizing the relationship between a categorical and numeric variable can be useful in many cases. This can be accomplished by using the hue mapping feature of the `histplot()` Seaborn function. In this case, we plotted the Grocery variable histogram with a different color for every category of the Channel and Region variables. As we can see, retail customers tend to spend more on grocery products compared to Horeca customers, as the histogram bins with large values belong to the Retail category. Furthermore, customers with high annual spendings come from the Lisbon & Other regions, as those from Oporto tend to spend less on grocery products, with a few exceptions.

```
fig, ax = plt.subplots(figsize = (12,8))

sns.boxplot(data=data[numeric], orient='h', ax = ax)
ax.set_xlim(0, 60000)

plt.show()
```



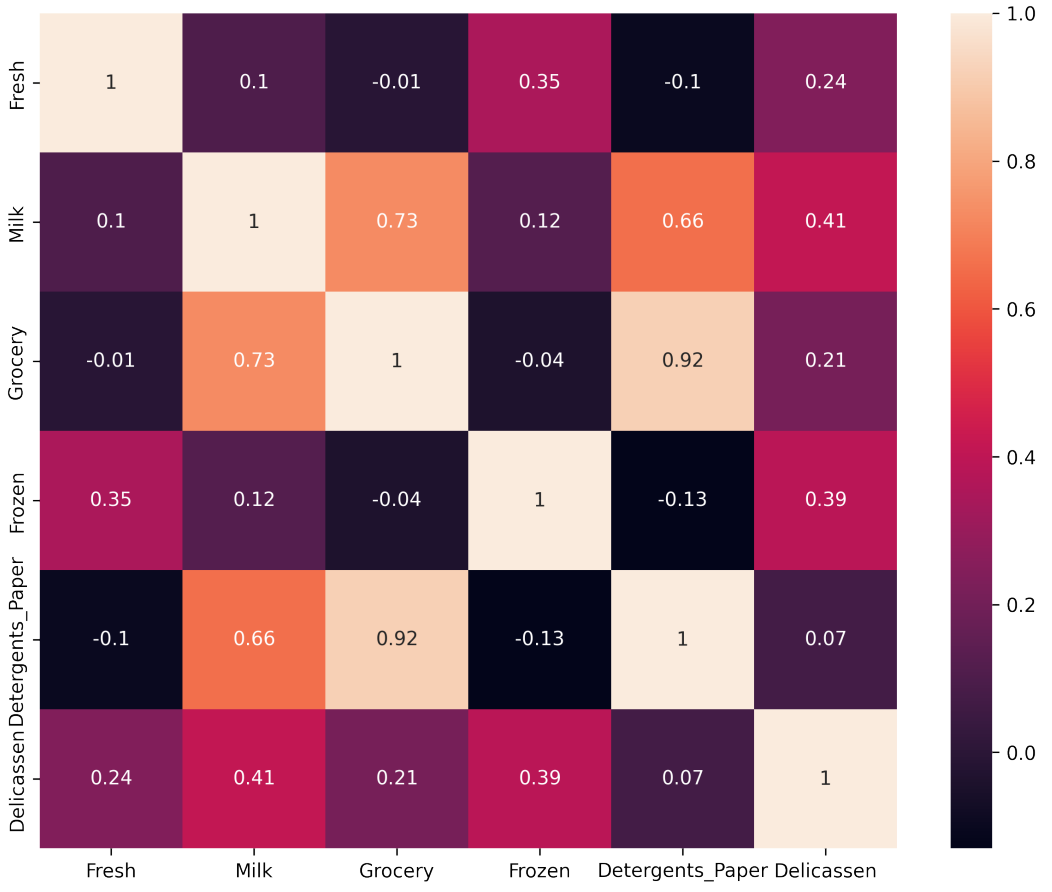
We use the `boxplot()` Seaborn function to visualize the quartiles of every numeric variable. Each box plot includes the min, Q1, median, Q3 and max values of the respective variable, as well as the outliers that are visualized as diamond symbols. As we can see, all numeric features have multiple outliers, but keep in mind that feature outliers are different from dataset outliers, the former referring to the each feature separately, instead of the dataset as a whole. Our goal in this chapter, is to take into account all features and identify the dataset outliers.

```
group = data.groupby(['Channel', 'Region']).size().unstack()
group.plot(kind='bar', figsize=(12,8))

plt.show()
```



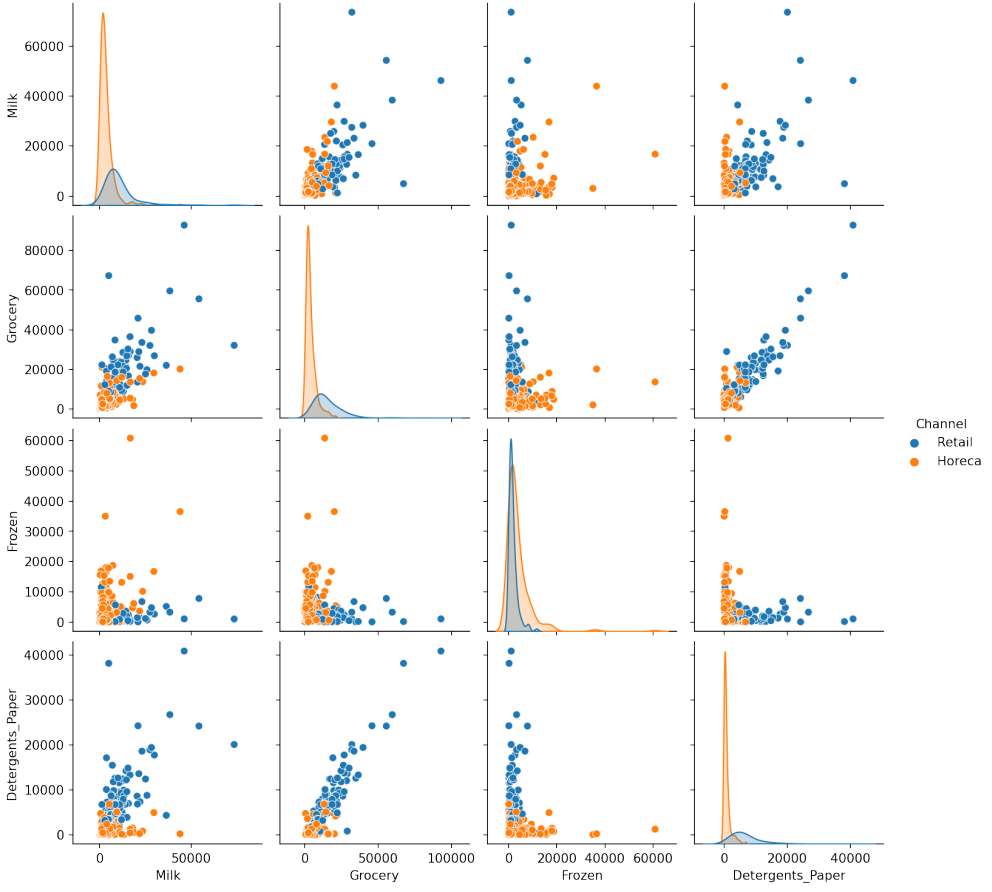
Visualizing the relationship between two categorical variables is often useful and informative. In this case, we use the `groupby()` pandas function to create a dataframe with the number of wholesale customers for each region, grouped by the channel they belong to. This is accomplished by using the aggregating function `size()` that helps us calculate group sizes. After doing that, we create a bar plot with the `plot()` pandas function. As we can see, the Lisbon and Other regions have significantly more wholesale customers belonging to the Horeca channel, in comparison to the Retail channel. The Horeca channel has more customers in Oporto as well, but the difference is smaller compared to the other two regions.

```
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr().round(decimals=2), annot=True)

plt.show()
```



We use the `heatmap()` Seaborn function to create a heatmap plot that visualizes the correlation values of all variable pairs. As we can see, some variable pairs have strong linear relationships, such as the Grocery and Detergents_Paper variables, with a correlation value of 0.92. Furthermore, the Grocery variable is also highly correlated with the Milk variable, having a correlation value of 0.73.

```
cols = ['Milk', 'Grocery', 'Frozen', 'Detergents_Paper']
sns.pairplot(data, vars=cols, hue='Channel')
plt.show()
```



We used the `pairplot()` Seaborn function to create a scatter plot matrix for the Wholesale Customers dataset, with the diagonal being the KDE plot of each variable. We also applied hue mapping to highlight the differences between the Retail and Horeca categories of the Channel variable. We can see that some of the variable pairs have strong linear relationships, as it was previously observed on the heatmap plot. Furthermore, there are some data points on each scatter plot that differ significantly from the rest, thus being outliers. Finally, the KDE plots indicate that customers from the Retail and Horeca channels have different spending habits.

Initializing the PyCaret Environment

```
anomaly = setup(data, session_id = 8477)
```

	Description	Value
0	session_id	8477
1	Original Data	(440, 8)
2	Missing Values	False
3	Numeric Features	6
4	Categorical Features	2
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(440, 11)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name
13	USI	7891

As in previous chapters, the `setup()` function is used to initialize the PyCaret environment, thus creating a pipeline that prepares the data for model training and deployment. This is a powerful function with numerous data preprocessing features, such as data imputation to deal with missing values, data transformation, data normalization and many others. In this case, the default function parameters are acceptable, so we aren't making any changes. If you want to read more details about the `setup()` function and its numerous features, please refer to the PyCaret Anomaly Detection module[3] documentation page.

[3] https://pycaret.readthedocs.io/en/latest/api/anomaly.html

Creating and Assigning the Model

```
model = create_model('lof', fraction = 0.05)

data_assigned = assign_model(model)
data_assigned.head(10)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Anomaly	Anomaly_Score
0	Retail	Other	12669	9656	7561	214	2674	1338	0	1.107687
1	Retail	Other	7057	9810	9568	1762	3293	1776	0	1.027102
2	Retail	Other	6353	8808	7684	2405	3516	7844	0	1.398439
3	Horeca	Other	13265	1196	4221	6404	507	1788	0	1.200384
4	Retail	Other	22615	5410	7198	3915	1777	5185	0	1.164052
5	Retail	Other	9413	8259	5126	666	1795	1451	0	1.184313
6	Retail	Other	12126	3199	6975	480	3140	545	0	1.130491
7	Retail	Other	7579	4956	9426	1669	3321	2566	0	1.013751
8	Horeca	Other	5963	3648	6192	425	1716	750	0	1.201904
9	Retail	Other	6006	11093	18881	1159	7425	2098	0	1.053333

We use the `create_model()` function to train the Local Outlier Factor model on the Wholesale Customers dataset. After doing that, we assign anomaly labels and scores to the dataset, by using the `assign_model()` function. As we can see, two columns have been added to the dataset, containing the anomaly label and score for each instance, as they were calculated by the model. Evidently, instances that have been flagged as inliers (anomaly = 0) have an anomaly score close to 1. As it was mentioned earlier, the Local Outlier factor model assigns anomaly scores significantly larger than 1 to outliers.

The `fraction` parameter of the `create_model()` function lets us change the estimated proportion of outliers in the dataset, with a default value of 5%. By setting a larger value to this parameter, instances with lower anomaly scores will be flagged as outliers as well. Finding the correct value for the `fraction` parameter can be challenging, as we rarely know the exact number of outliers in a dataset. Regardless, experimenting with different values and evaluating the results can be helpful in most cases.

```
data_inliers = data_assigned.query('Anomaly == 0')
data_outliers = data_assigned.query('Anomaly == 1')

data_outliers.head(10)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Anomaly	Anomaly_Score
23	Retail	Other	26373	36423	22019	5154	4337	16523	1	2.249573
47	Retail	Other	44466	54259	55571	7782	24171	6465	1	3.224197
61	Retail	Other	35942	38369	59598	3254	26701	2017	1	2.799411
65	Retail	Other	85	20959	45828	36	24231	1423	1	2.120687
71	Horeca	Other	18291	1266	21042	5373	4173	14472	1	2.273296
85	Retail	Other	16117	46197	92780	1026	40827	2944	1	3.970515
86	Retail	Other	22925	73498	32114	987	20070	903	1	3.435620
93	Horeca	Other	11314	3090	2062	35009	71	2698	1	3.421567
125	Horeca	Other	76237	3473	7102	16538	778	918	1	2.366692
145	Retail	Other	22039	8384	34792	42	12591	4430	1	1.984799
181	Horeca	Other	112151	29627	18148	16745	4948	8550	1	4.716803
183	Horeca	Other	36847	43950	20170	36534	239	47943	1	4.267115
211	Retail	Lisbon	12119	28326	39694	4736	19410	2870	1	1.730137
254	Horeca	Lisbon	10379	17972	4748	4686	1547	3265	1	1.859050
265	Horeca	Lisbon	5909	23527	13699	10155	830	3636	1	1.936609

We use the `query()` pandas function to create two additional dataframes, one containing the inlier instances only, and one with the outliers. After doing that, we print the first rows of the outliers dataset, by using the `head()` pandas function. As expected, all the instances are flagged as outliers in the anomaly column. Furthermore, we can see that all the anomaly scores are significantly higher than 1, indicating that those instances have been correctly identified as outliers by the Local Outlier Factor model.

Evaluating the Model

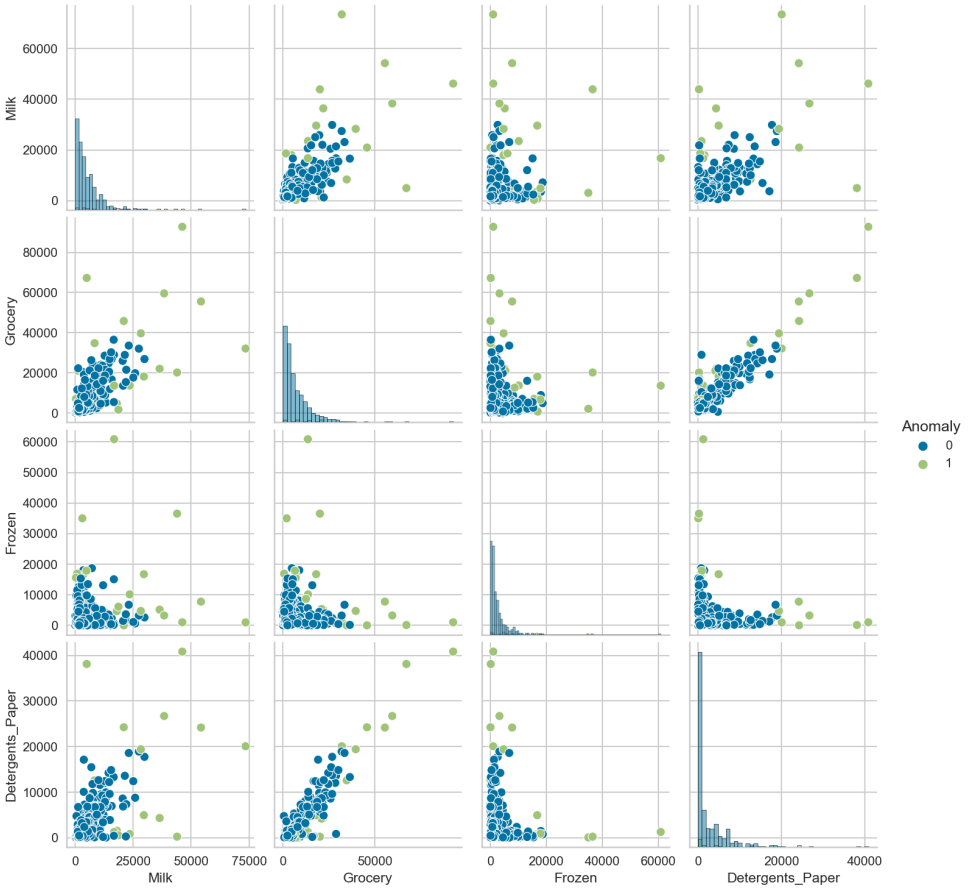
```
data_skew = data[numeric].skew()
inliers_skew = data_inliers[numeric].skew()

print("Skewness of original dataset")
print(data_skew.round(decimals=2).to_string())
print("Mean skewness        %0.2f" %data_skew.mean())

print("\nSkewness of inlier dataset")
print(inliers_skew.round(decimals=2).to_string())
print("Mean skewness       %0.2f" %inliers_skew.mean())
```

```
Skewness of original dataset
Fresh               2.56
Milk                4.05
Grocery             3.59
Frozen              5.91
Detergents_Paper    3.63
Delicassen         11.15
Mean skewness       5.15

Skewness of inlier dataset
Fresh              1.61
Milk               1.98
Grocery            1.63
Frozen             2.25
Detergents_Paper   2.09
Delicassen         3.41
Mean skewness      2.16
```

We used the `skew()` pandas function to calculate the skewness of numeric features for the original dataset, as well as the inlier dataset. As we can see, the original dataset features are extremely skewed, especially the Frozen and Delicassen variables. On the other hand, the inlier dataset features are signficantly less skewed, with the mean skewness value dropping from 5.15 to 2.16. This indicates that the anomaly detection model identified most of the outlier values successfully.

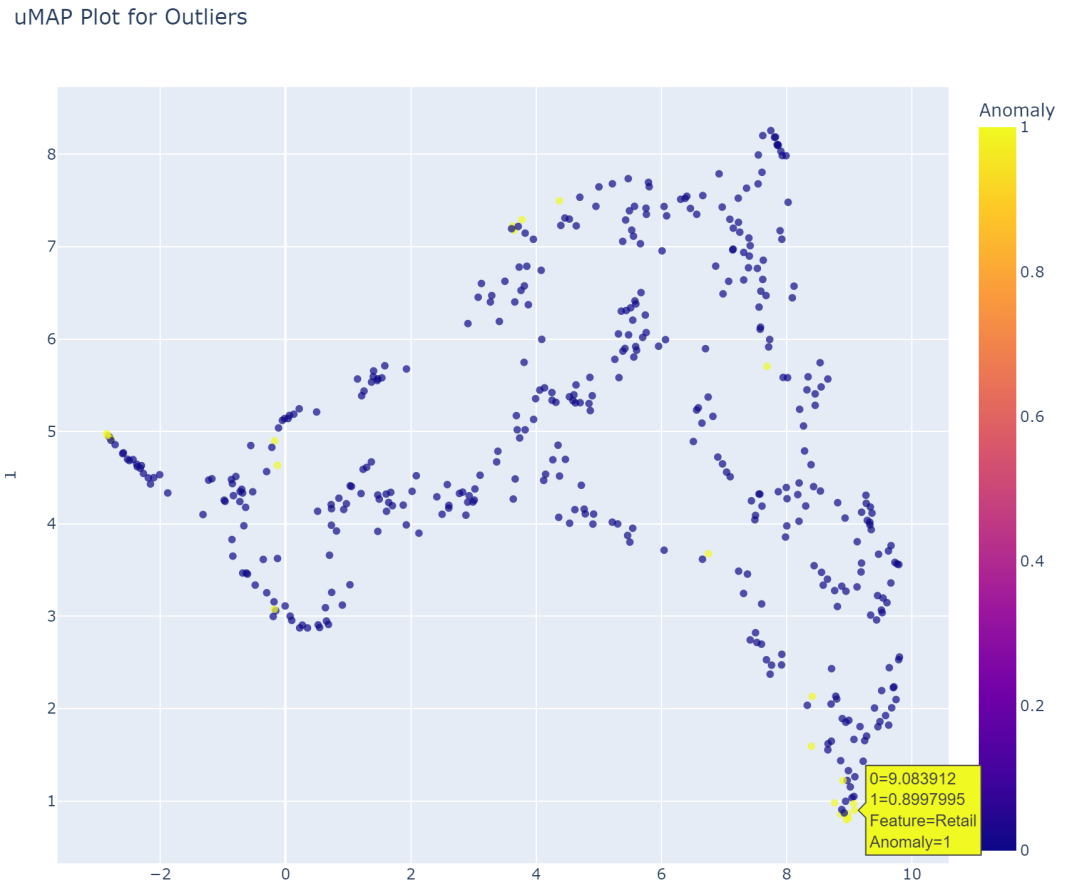
Plotting the Model

```
cols = ['Milk', 'Grocery', 'Frozen', 'Detergents_Paper']

sns.pairplot(data_assigned, vars=cols, hue='Anomaly', diag_kind = 'hist')
plt.show()
```



We used the `pairplot()` Seaborn function to create another scatter plot matrix for the Wholesale Customers dataset. In this case, we applied hue mapping to highlight the instances that have been flagged as outliers. As we can see in each scatter plot, most of the instances that differ significantly from the rest, have been successfully identified as outliers (Anomaly = 1).

```
plot_model(model, 'umap')
```



The `plot_model()` function can be used to plot various useful graphs for our anomaly detection model. In this case, we created a Uniform Manifold Approximation and Projection (UMAP) plot, a powerful dimensionality reduction technique that can be used for visualization purposes[34], similarly to t-SNE[35]. UMAP is based in Riemannian geometry and algebraic topology, providing a scalable algorithm that applies to real-world data. As we can see, some of the outlier instances have formed a cluster at the bottom right corner of the plot, highlighting the fact that they have similar characteristics to each other.

Saving the Model

```
save_model(model, 'anomaly_detection_model')
```

As in previous chapters, we use the `save_model()` function to save our anomaly detection model to the local disk, for future use. The model is saved as a serialized pickle file, that can be loaded with the complementary `load_model()` function. By default, the entire transformation pipeline is stored along with the model, but we can avoid that by setting the `model_only` parameter to `True`.

6. Natural Language Processing

Natural language processing (NLP) is placed at the intersection of computational linguistics and machine learning. The main goal in this dynamic field is to extract information and insights from natural languages, i.e. those spoken by humans in their every day lives. NLP comprises a wide variety of methods and techniques, including topic modelling, sentiment analysis, machine translation, document summarization and speech-to-text conversion. In this chapter, we are going to focus on topic modelling, as it is supported by the NLP module of the PyCaret library. We can use this technique to discover topics, i.e. the hidden structures that let us semantically group a collection of documents, known as the corpus. Latent Dirichlet Allocation[36] is a generative probabilistic model that can be used for topic modelling, and it is defined in the following equation.

$$p(\mathbf{w} \mid \boldsymbol{\alpha}, \boldsymbol{\beta}) = \int p(\theta \mid \alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p\left(z_n \mid \theta\right) p\left(w_n \mid z_n, \boldsymbol{\beta}\right) \right) d\theta$$

- $p(\mathbf{w} \mid \boldsymbol{\alpha}, \boldsymbol{\beta})$ is the marginal distribution of a document.
- $\mathbf{w}$ is a set of N words, known as a document.
- $p(\theta \mid \alpha)$ is the Dirichlet distribution of topic mixtures.
- w_n is the nth word of the document.

Natural Language Processing with PyCaret

Besides Latent Dirichlet Allocation, there are numerous other topic modelling algorithms, including Latent Semantic Indexing[37], Hierarchical Dirichlet Process[38], Random Projections[39] and Non-Negative Matrix Factorization[40]. In the rest of this chapter, we are going to apply topic modelling to a collection of documents (corpus), by using the PyCaret NLP module. Afterwards, those topic probabilities will be used as features to train a classification model on the dataset.

Downloading the Additional Resources

Before we begin working on the project, we need to download some additional resources, because the PyCaret NLP module requires the Spacy English language model, as well as the TextBlob corpora. These aren't included with the PyCaret installation, so you have to download them manually, by executing the following commands on your Anaconda terminal.

```
python -m spacy download en_core_web_sm
python -m textblob.download_corpora
```

After doing that, the additional resources are going to be downloaded and you'll be able to use the natural language processing module without any problems. Obviously, if you have already downloaded those files for another project that utilizes the Spacy and TextBlob libraries, you can safely skip this step, and move on to the next chapter section.

Importing the Necessary Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from wordcloud import WordCloud
from spacy.lang.en.stop_words import STOP_WORDS
from pycaret import nlp
from pycaret import classification
mpl.rcParams['figure.dpi'] = 300
```

We begin by importing the libraries that are necessary for our project, including pandas, Matplotlib, WordCloud, as well as the NLP and Classification modules of PyCaret. We also import a list of the english language stopwords from the spaCy library, as it will be useful in the EDA section. You shouldn't worry if you aren't familiar with stopwords, as they will be explained later, along with other common NLP terms and concepts. Finally, we set the Matplotlib figure DPI to 300, so we get high-resolution plots for this book.

Loading the Dataset

In this chapter, we are going to use the BBC News dataset[1] that was created by researchers from the University College Dublin (UCD), and provided freely for educational and scientific purposes. This dataset includes more than 2000 articles from the BBC News[2] website, that belong to 5 distinct classes, i.e. business, entertainment, politics, sport and tech. The BBC News dataset can be used for general Natural Language Processing tasks, as well as multiclass classification, making it an optimal choice for the needs of this project. You can access it by cloning the official book repository[3], or download it directly by visiting this link.

[1] http://mlg.ucd.ie/datasets/bbc.html
[2] https://www.bbc.com/news
[3] https://github.com/derevirn/pycaret-book

```
data = pd.read_csv('bbc-text.csv')
data.head(10)
```

	category	text
0	tech	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	sport	tigers wary of farrell gamble leicester say ...
3	sport	yeading face newcastle in fa cup premiership s...
4	entertainment	ocean s twelve raids box office ocean s twelve...
5	politics	howard hits back at mongrel jibe michael howar...
6	politics	blair prepares to name poll date tony blair is...
7	sport	henman hopes ended in dubai third seed tim hen...
8	sport	wilkinson fit to face edinburgh england captai...
9	entertainment	last star wars not for children the sixth an...

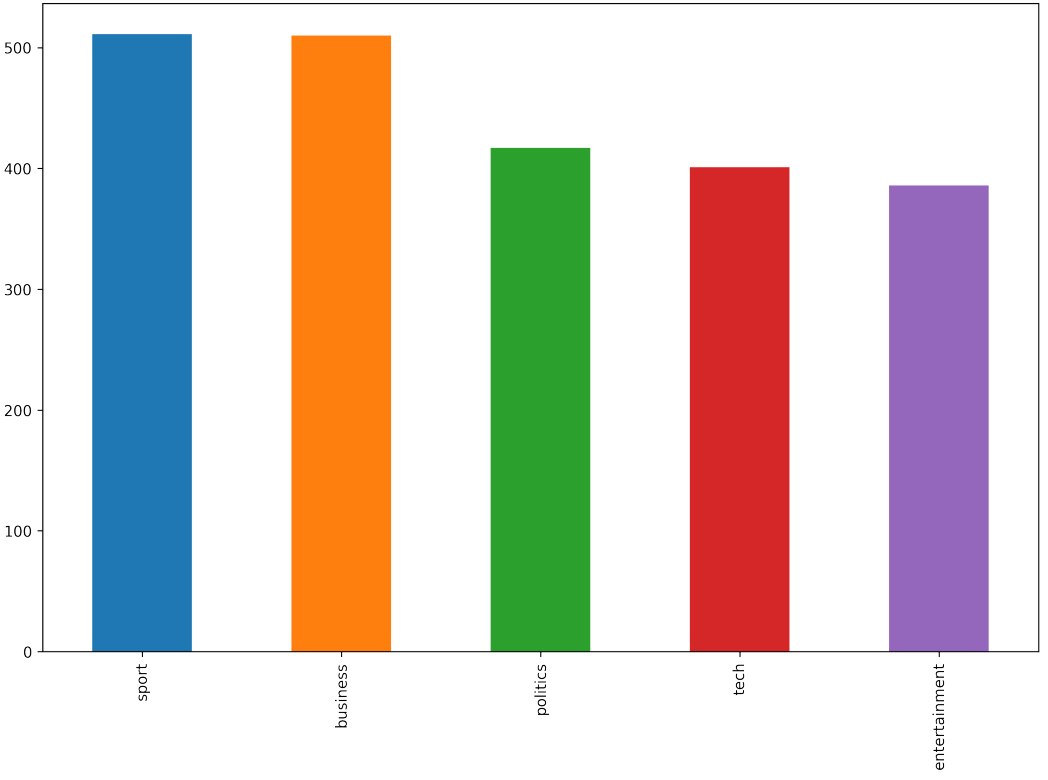
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   category  2225 non-null   object
 1   text      2225 non-null   object
dtypes: object(2)
```

We use the `read_csv()` pandas function to load the dataset to a dataframe. As we can see, there are two columns, including the category, as well as the text content of each article. After doing that, we print some basic info about the dataset, by using the `info()` function. As noted earlier, there are 2225 articles, that have zero missing values for both columns.

Exploratory Data Analysis

```
color = ['C0', 'C1', 'C2', 'C3', 'C4']
categories = data['category'].value_counts()
categories.plot(kind = 'bar', figsize = (12,8), color = color)
plt.show()
```

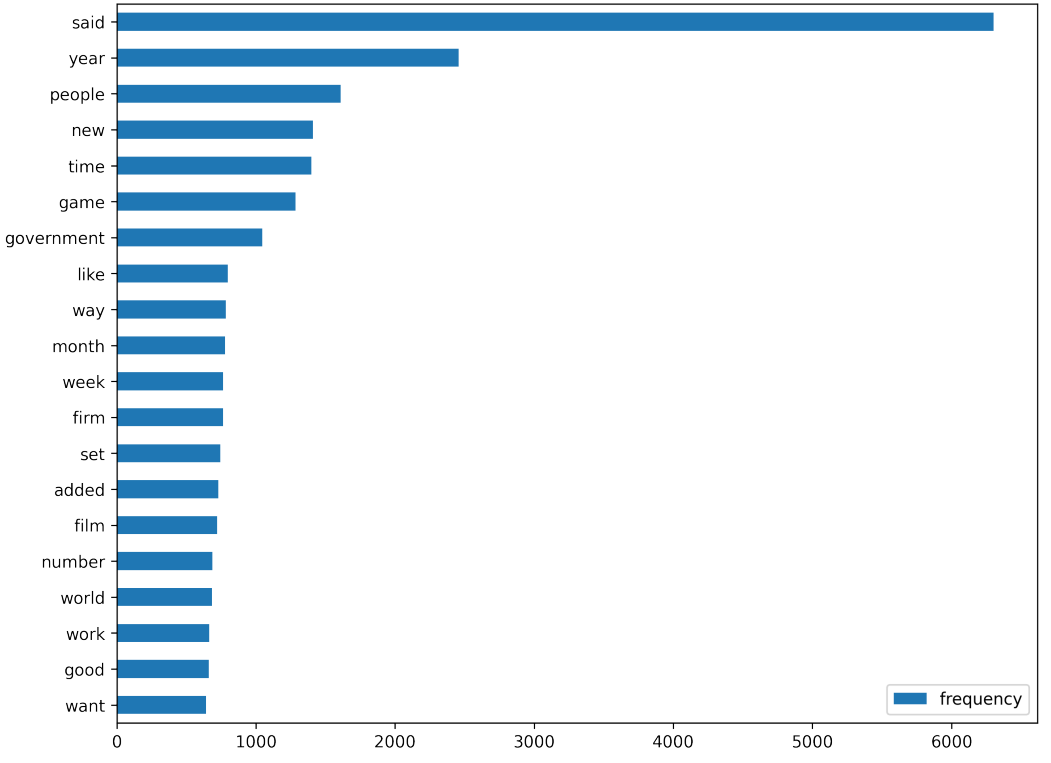


We are now going to perform Exploratory Data Analysis (EDA) on the BBC News dataset. By using the `value_counts()` and `plot()` pandas functions, we are able to create a bar chart that visualizes class proportions. As we can see, the dataset is imbalanced because the classes aren't evenly distributed. We are going to deal with this issue later, as it may cause problems with classification model training. As it is evident, the two most common categories are sport and business, each having about 500 articles. On the other hand, entertainment is the least popular, having fewer than 400 articles.

```
wc = WordCloud(width = 1800, height = 1200, stopwords = STOP_WORDS,
     background_color = 'white', min_word_length = 3, max_words = 100)

data_tech = data.query(" category == 'tech' ")['text']
text_tech = ' '.join(data_tech.to_list())
wc_img = wc.generate(text_tech)

plt.figure(figsize = (12,8))
plt.imshow(wc_img, interpolation = 'bilinear')
plt.axis("off")
plt.show()
```

The word frequencies of a given text can be easily visualized with a word cloud. This type of visualization displays every word with a different font size and color, depending on the number of its occurrences, thus highlighting the prominence of common words. In this case, we visualized the tech category of the BBC News dataset, by using the WordCloud Python library. As expected, words that are related to technology are highlighted as the most prominent.

```
text = ' '.join(data['text'].to_list())
freq = wc.process_text(text)
df_freq = pd.DataFrame.from_dict(freq, orient='index', columns=['frequency'])
df_freq = df_freq.sort_values('frequency')
df_freq[-20:].plot(kind = 'barh', figsize = (10,8))

plt.show()
```



Besides plotting a word cloud, we can also visualize word frequencies with a simple bar chat. In this case, we used the `process_text()` WordCloud function to extract word frequencies from the entire BBC News dataset, in the form of a Python dictionary. After doing that, we created a pandas datafame based on that dictionary. Finally, we plotted the word frequencies as a horizontal bar chart.

Initializing the NLP Environment

```
nlp_ = nlp.setup(data = data, target='text', session_id = 6842)
data_ = nlp.get_config('data_')
data_.head(10)
```

Description	Value
session_id	6842
Documents	2225
Vocab Size	15625
Custom Stopwords	False

	category	text
0	tech	tv future hand viewer system plasma digital_vi...
1	business	leave book alone former accuse oversee fraud n...
2	sport	gamble leicester say rush make bid decide swit...
3	sport	yeade face premiership side leader yeade fa_cu...
4	entertainment	ocean_twelve raid ocean_twelve crime caper go ...
5	politics	hit back say claim act attack show rattle oppo...
6	politics	name likely name may election day parliament r...
7	sport	end third seed slump straight set defeat rain ...
8	sport	make return play injure take part full contact...
9	entertainment	last star_war child sixth final movie may suit...

We are now going to initialize the PyCaret NLP environment and create the transformation pipeline, by using the `setup()` function. The `target` parameter lets us specify the text column of the dataset, that will go through a number of preprocessing steps, as described in the next page. After this process is completed, the first 10 instances of the preprocessed dataset are printed.

Numeric & Special Character Removal

Numbers and punctuation are not informative in the context of natural language processing, so PyCaret removes all numeric and special characters from the corpus. Those unnecessary characters are replaced with spaces by using regular expressions.

Word Tokenization

Tokenization is the process of splitting the corpus into tokens, i.e. smaller units which are usually words. This is a fundamental part of natural language processing that is typically one of the first steps in the pipeline, as it must be applied before any further analysis.

Stopword Removal

Stopwords are words that are automatically removed from the text because they are very common, thus providing no useful information. Common stopwords include articles, pronouns and prepositions. Furthermore, the given corpus may also include specific additional words that appear too often without being informative. You can easily deal with that problem by assigning a list of words to the `custom_stopwords` parameter of the `setup()` function, that will subsequently be removed from the corpus.

Bigram & Trigram Extraction

n-grams are sequences of n consecutive tokens that are used in natural language processing for a variety of tasks, including spelling error correction, text prediction, text mining, statistical analysis and others. The PyCaret NLP module extracts all the n-grams of the given corpus for $n = 2$ and $n = 3$. In those special cases, they are known as bigrams and trigrams respectively.

Lemmatization

The main form of a word as it appears in the dictionary is known as a lemma. Usually, words are inflected to express different grammatical functions, such as tense and case. Lemmatization is the process of grouping together the inflected forms of a word, so they can be identified by its lemma.

Creating and Assigning the Topic Model

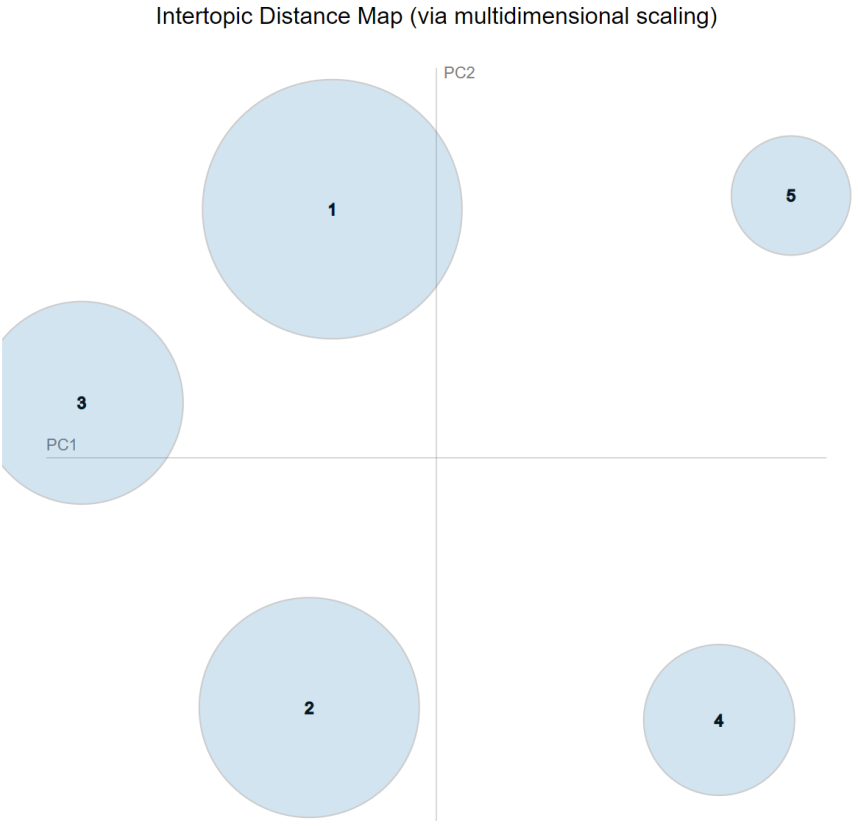
```
cols = ['Topic_0','Topic_1','Topic_2',
        'Topic_3','Topic_4','category']

lda = nlp.create_model('lda', num_topics = 5)
data_assigned = nlp.assign_model(lda)
data_assigned_ = data_assigned[cols]

data_assigned_.head(10)
```

	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4	category
0	0.060324	0.831212	0.063175	0.011273	0.034016	tech
1	0.115085	0.039387	0.001271	0.464457	0.379799	business
2	0.584945	0.004143	0.001588	0.002269	0.407055	sport
3	0.876524	0.004787	0.001918	0.002714	0.114057	sport
4	0.237407	0.005586	0.677938	0.047968	0.031100	entertainment
5	0.091791	0.001754	0.016693	0.000936	0.888826	politics
6	0.076346	0.003809	0.001508	0.002160	0.916178	politics
7	0.740604	0.005800	0.154086	0.003107	0.096402	sport
8	0.744063	0.009129	0.003764	0.005489	0.237555	sport
9	0.100310	0.004580	0.512188	0.002494	0.380428	entertainment

The `create_model()` function can be used to easily train a topic model on the preprocessed dataset. In this case, we train a Latent Dirichlet Allocation model with 5 topics, one for each of the dataset classes. After doing that, we use the `assign_model()` function that assigns the topic proportions to the dataset, as well as the dominant topic. Finally, we create a new dataset containing the topic proportions as well as the news article class, and print the first 10 instances. As we can see, the sport and politics classes correspond to topics 0 and 4 respectively, indicating that the model has successfully detected the topics of each dataset class.

Plotting the Topic Model

```
nlp.plot_model(model = lda, plot = 'topic_model')
```



The `plot_model()` function lets us visualize the topic model by creating various types of plots. Furthermore, in case we don't pass a model parameter to the function, the preprocessed dataset will be visualized instead. In this case, we create a plot that is known as the intertopic distance map[41], based on the pyLDAvis[4] library. Each topic is represented by a circle, with its diameter being proportional to the number of instances that belong to the topic. The distance between circles indicates the topic similarity, or lack thereof. In this case, none of the circles seem to be overlapping with each other, suggesting that our topic model performs well.

[4] https://pyldavis.readthedocs.io/en/latest/readme.html

Initializing the Classification Environment

```
classf = classification.setup(data_assigned_, target = 'category',
         fix_imbalance = True, train_size = 0.8, session_id = 3100)
```

	Description	Value
0	session_id	3100
1	Target	category
2	Target Type	Multiclass
3	Label Encoded	business: 0, entertainment: 1, politics: 2, sport: 3, tech: 4
4	Original Data	(2225, 6)
5	Missing Values	False
6	Numeric Features	5
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(1780, 5)
12	Transformed Test Set	(445, 5)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False

We now have a dataset with 5 numeric features and a class label, thus enabling us to train a classification model on it. By doing that, we'll be able to create a text classification pipeline, using the topic and classification models. At this point, we are dealing with a multiclass classification task, so the process will not be examined in detail, as it has been covered in a previous chapter. First of all, we initialize the PyCaret classification environment, that prepares the dataset for model training. During Exploratory Data Analysis, we discovered that the dataset is imbalanced, so we set the `fix_imbalance` parameter to True.

Creating the Classification Model

```
catboost = classification.create_model('catboost')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8933	0.9871	0.8936	0.8943	0.8934	0.8661	0.8662
1	0.9607	0.9955	0.9603	0.9619	0.9609	0.9506	0.9508
2	0.9326	0.9951	0.9299	0.9371	0.9335	0.9154	0.9161
3	0.9157	0.9931	0.9185	0.9200	0.9154	0.8945	0.8957
4	0.9101	0.9906	0.9057	0.9151	0.9102	0.8871	0.8882
5	0.9213	0.9934	0.9236	0.9262	0.9218	0.9016	0.9025
6	0.8652	0.9833	0.8609	0.8808	0.8678	0.8304	0.8329
7	0.8933	0.9887	0.8938	0.8988	0.8946	0.8661	0.8668
8	0.9045	0.9889	0.8990	0.9050	0.9043	0.8801	0.8803
9	0.8820	0.9895	0.8887	0.8919	0.8815	0.8525	0.8553
Mean	0.9079	0.9905	0.9074	0.9131	0.9084	0.8844	0.8855
SD	0.0256	0.0036	0.0258	0.0230	0.0253	0.0322	0.0317

The `create_model()` function lets us easily train a classification model on the dataset. In this case, we are going to create a Catboost model, as it generally performs well in classification tasks. As we can see, the trained Catboost model manages to achieve a mean accuracy of 90.79%, which is very satisfactory. Of course, we can also compare the performance of all classification models by using the `compare_models()` function, but this is beyond the scope of this chapter. As mentioned earlier, if you want a more detailed overview of the PyCaret classification module, you can refer to the associated book chapter.

Finalizing and Saving the Models

```
catboost_final = classification.finalize_model(catboost)

nlp.save_model(lda, 'lda_model')
classification.save_model(catboost_final, 'catboost_model')
```

The `setup()` function of the PyCaret classification module splits the dataset into train and test subsets. It is therefore advised to use the `finalize_model`() function, so the Catboost model is trained on the complete dataset, thus utilizing the test subset too. After doing that, we use the `save_model()` function to save both the classification model, as well as the topic model to the local storage. The models are saved as serialized pickle files that can be loaded with the complementary `load_model()` functions.

7. Deploying a Machine Learning Model

After training a PyCaret model that performs well and is sufficiently accurate, you will probably consider putting it to production. After all, running machine learning experiments may be interesting, but typically our goal is to develop a useful product or service that provides value to users. There are numerous ways to utilize a machine learning model, the most common being its deployment as a web application or Application Programming Interface (API).

The Python ecosystem provides numerous tools and libraries that can help us deploy a model, including a variety of web application frameworks. For example, Django[1] is a high-level web framework that can be used to quickly develop applications, while being secure and scalable. Another popular library is Flask[2], considered to be a micro framework because of its minimal core functionality, that can be extended as needed with a multitude of libraries. Alternatively, you can also deploy your machine learning model as a REST API, by using a specialized framework, such as the powerful FastAPI[3] library. After developing an application or API, the next step is to deploy it on a cloud computing service. There are numerous available, the most popular being Amazon Web Services, Microsoft Azure and Google Cloud. Those services provide numerous advanced features, including virtual machines, support for containerized apps with Kubernetes, cloud storage and many others.

In this chapter, our goal is learning how to develop machine learning web applications with the Streamlit[4] framework, by focusing on two case studies. The first is based on the regression model that we trained in chapter 2, while the second is utilizing the classification model of chapter 3. After developing those applications, we are going to easily deploy them, by using the Streamlit Cloud service.

[1] https://www.djangoproject.com/
[2] https://flask.palletsprojects.com/
[3] https://fastapi.tiangolo.com/
[4] https://streamlit.io/

The Streamlit Framework

Streamlit is a Python framework that is designed to enable the effortless development of machine learning and data science web applications. It is an excellent choice for beginners, as it has numerous powerful features, while being easy to use. Streamlit includes a variety of widgets, such as radio buttons, checkboxes, sliders and text inputs. Furthermore, it supports several visualization libraries that can be used to plot pandas dataframes, such as Matplotlib, Vega Lite and deck.gl. As a result of those features, developing a machine learning application based on Streamlit is particularly convenient.

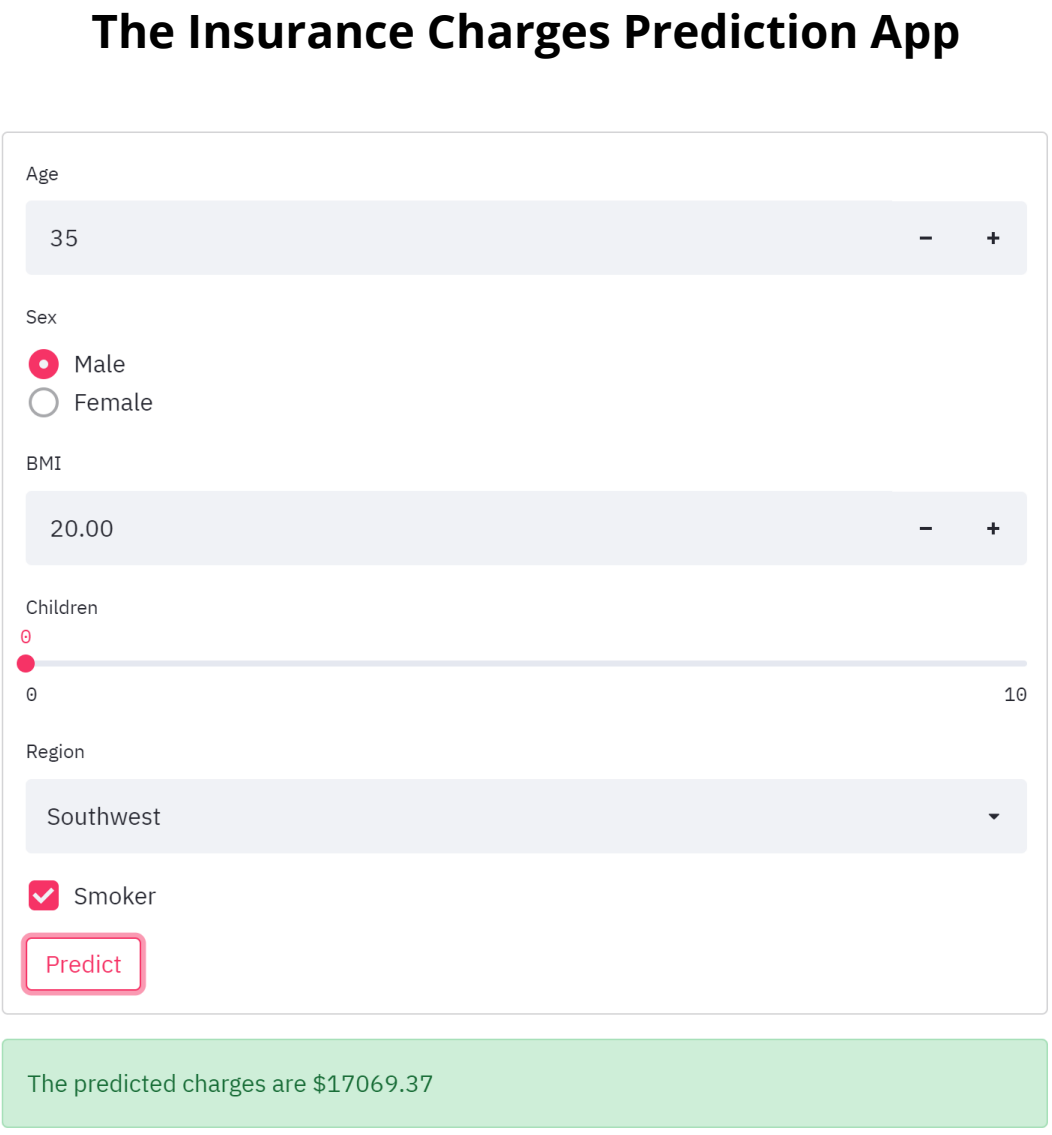
Streamlit

After developing a web application with Streamlit, you can easily deploy it on the cloud computing service of Streamlit Inc, aptly named Streamlit Cloud[5]. This service provides numerous advanced features, such as one-click deployment of Github repositories, live updates when you push code changes, user authentication with OAuth or single sign-on (SSO) and others. The Community tier of Streamlit Cloud is freely available for everybody, but some of the features are restricted to the premium tiers that are available for a monthly fee.

The Streamlit Python library can be easily installed by executing the following command on your Anaconda terminal.

```
pip install streamlit
```

[5] https://streamlit.io/cloud



In the above screenshot, we can see the interface of the Insurance Charges Prediction App. Users can enter their personal information in the form, and get a prediction of the insurance charges that they are going to be billed. A variety of Streamlit widgets was used, such as radio button, slider and select box, depending on the type of each variable. As mentioned earlier, the application is based on the Gradient Boosting Regressor model that we trained in chapter 2.

Developing the Web Application

We are now going to examine the Insurance Charges Prediction App source code, contained in the `app.py` file. Keep in mind that the following code is not part of a Jupyter notebook, so it should be edited with the IDE of your preference, instead of JupyterLab. I suggest using Spyder, a free IDE that is included with Anaconda, or Microsoft VS Code.

```
import pandas as pd
import streamlit as st
from pycaret.regression import load_model, predict_model
```

We begin by importing the necessary libraries, including Streamlit, pandas and the `load_model()` and `predict_model()` functions of the PyCaret Regression module. Those functions will let us load the stored regression model, as well as make predictions on user input with it.

```
st.set_page_config(page_title="Insurance Charges Prediction App")

@st.cache(allow_output_mutation=True)
def get_model():
    return load_model('regression_model')

def predict(model, df):
    predictions = predict_model(model, data = df)
    return predictions['Label'][0]

model = get_model()
```

We set the page title by using the `set_page_config()` Streamlit function. After doing that, we define the `get_model()` function which is a wrapper for the PyCaret `load_model()`. The reason for creating this function is to enable the Streamlit caching mechanism, by putting the `@st.cache` decorator before defining the function. Doing this will improve performance, as the PyCaret model will be loaded from the Streamlit cache every time the function is called. Additionally, we define the `predict()` function which uses the model to make predictions on a pandas dataframe and returns the result. We then load the regression model to a variable named model.

```
st.title("Insurance Charges Prediction App")
st.markdown("Enter your personal details to get a prediction of your insurance\
        charges. This is a simple app showcasing the abilities of the PyCaret\
        regression module, based on Streamlit. For more information visit the\
        [Simplifying Machine Learning with PyCaret]\
        (https://leanpub.com/pycaretbook/) book website.")
```

We create a heading, as well as a brief description for the application, by using the `title()` and `markdown()` functions. As the name suggests, the `markdown()` function lets you display text that is formatted in the Markdown markup language. Furthermore, it also supports LaTeX expressions, as well as HTML. Keep in mind that adding HTML code to a Streamlit application is discouraged for security reasons, so it has to be enabled by setting the `unsafe_allow_html` parameter to `True`.

```
form = st.form("charges")
age = form.number_input('Age', min_value=1, max_value=100, value=25)
sex = form.radio('Sex', ['Male', 'Female'])
bmi = form.number_input('BMI', min_value=10.0, max_value=50.0, value=20.0)
children = form.slider('Children', min_value=0, max_value=10, value=0)
region_list = ['Southwest', 'Northwest', 'Northeast', 'Southeast']
region = form.selectbox('Region', region_list)
if form.checkbox('Smoker'):
    smoker = 'yes'
else:
    smoker = 'no'

predict_button = form.form_submit_button('Predict')
```

A form containing the widgets that are related to user input is created, by using the `form()` function. After doing that, we then create a widget for every variable that is necessary to make a prediction with our regression model. The type of widget depends on the variable, e.g. a number input is appropriate for the age and bmi variables, while a checkbox is best for the binary smoker variable. Finally, we create the form submission button, by using the `form_submit_button()` function. If you want a list of all the available Streamlit widgets, please refer to the API Reference[6].

[6] https://docs.streamlit.io/en/stable/api.html

```
input_dict = {'age' : age, 'sex' : sex.lower(), 'bmi' : bmi, 'children' : child\
ren,
               'smoker' : smoker, 'region' : region.lower()}
input_df = pd.DataFrame([input_dict])

if predict_button:
    out = predict(model, input_df)
    st.success('The predicted charges are ${:.2f}'.format(out))
```

The input_dict variable is a Python dictionary containing all user input that was submitted to the form. We use the `DataFrame()` constructor to instantiate a pandas dataframe object, based on the user input dictionary. Finally, when a user clicks the form button, we pass the dataframe to the `predict()` function and print the predicted charges, by using the `success()` function.

Running the Web Application Locally

We can run a Streamlit application locally, by executing the following command on the Anaconda terminal. A web server will be started, and the application will load automatically on the default web browser of our operating system.

```
streamlit run app.py

2021-09-22 23:11:56.984 INFO    numexpr.utils: Note: NumExpr detected 16 cores \
but "NUMEXPR_MAX_THREADS" not set, so enforcing safe limit of 8.
2021-09-22 23:11:56.993 INFO    numexpr.utils: NumExpr defaulting to 8 threads.

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.2:8501

2021-09-22 23:12:00.045 Initializing load_model()
2021-09-22 23:12:00.046 load_model(model_name=regression_model, platform=None, \
authentication=None, verbose=True)
Transformation Pipeline and Model Successfully Loaded
```

In this screenshot, we can see the Iris Classification App, which is the second example of this chapter. Users can enter the dimensions of an Iris plant, and get the predicted species as output. As mentioned before, this application is based on the Linear Discriminant Analysis model that we trained in chapter 3.

Developing the Web Application

We are now going to examine the source code of the Iris Classification App, included in the `app.py` file. Contrary to previous chapters, we aren't using Jupyter notebooks, so make sure to edit the file with a Python IDE instead.

```
import pandas as pd
import streamlit as st
from pycaret.classification import load_model, predict_model
```

As we did in the first example of this chapter, we start by importing the necessary libraries, including pandas and Streamlit. Furthermore, we also import the `load_model()` and `predict_model()` functions of the PyCaret Classification module, that will let us load the classification model and make predictions with it.

```
st.set_page_config(page_title="Iris Classification App")

@st.cache(allow_output_mutation=True)
def get_model():
    return load_model('classification_model')

def predict(model, df):
    predictions = predict_model(model, data = df)
    return predictions['Label'][0]

model = get_model()
```

We use the `set_page_config()` function to set a title for the Streamlit application, similarly to creating a `<title>` tag in a simple HTML page. After doing that, we define the function `get_model()` that wraps the `load_model()` function. We also enable the Streamlit caching mechanism, by adding the `@st.cache` decorator before the function is defined, so we can improve performance and loading times. The `predict()` function uses the classification model to make predictions on a dataframe, and afterwards returns the result.

```
st.title("Iris Classification App")
st.markdown("Choose the values for each attribute of the Iris plant that you\
        want to be classified. This is a simple app showcasing the abilities\
        of the PyCaret classification module, based on Streamlit. For more\
        information visit the [Simplifying Machine Learning with PyCaret]\
        (https://leanpub.com/pycaretbook/) book website.")
```

We use the `title()` and `markdown()` functions to create a heading and a description for the application. The `markdown()` function supports text that is formatted in Markdown, as well as LaTeX and HTML, although the latter is discouraged due to security concerns. Furthermore, LaTeX expressions should be wrapped in "$", so they can be displayed properly.

```
form = st.form("species")
sepal_length = form.slider('Sepal Length', min_value=0.0, max_value=10.0,
                           value=0.0, step = 0.1, format = '%f')
sepal_width = form.slider('Sepal Width', min_value=0.0, max_value=10.0,
                           value=0.0, step = 0.1, format = '%f')
petal_length = form.slider('Petal Length', min_value=0.0, max_value=10.0,
                           value=0.0, step = 0.1, format = '%f')
petal_width = form.slider('Petal Width', min_value=0.0, max_value=10.0,
                           value=0.0, step = 0.1, format = '%f')

predict_button = form.form_submit_button('Predict')
```

We create a form containing all the necessary input widgets, by using the `form()` function. We only used slider widgets, as the Iris plant variables are all numeric, and have a limited range. Furthermore, we set the `format` parameter to `%f`, so only one decimal place is displayed for each floating point number. Finally, we create a form submission button that is labelled **Predict**, by using the `form_submit_button()` function.

```
input_dict = {'sepal_length' : sepal_length, 'sepal_width' : sepal_width,
              'petal_length' : petal_length, 'petal_width' : petal_width}

input_df = pd.DataFrame([input_dict])

if predict_button:
    out = predict(model, input_df)
    st.success(f'The predicted species is {out}.')
```

We instantiate a pandas dataframe object with the `DataFrame()` constructor, based on a Python dictionary that contains all the data that was submitted to the form. When the form submission button is clicked, the dataframe is passed to the `predict()` function, and the result is displayed with the `success()` function.

Running the Web Application Locally

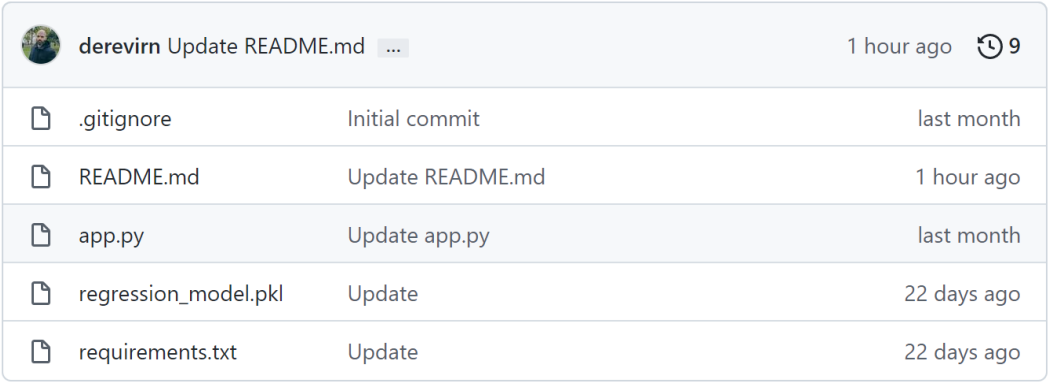
We can run the Iris Classification App locally, by executing the following command on the Anaconda terminal. After the web server is started, the application will load automatically on the default web browser.

```
streamlit run app.py

2021-09-23 03:51:14.819 INFO    numexpr.utils: Note: NumExpr detected 16 cores \
but "NUMEXPR_MAX_THREADS" not set, so enforcing safe limit of 8.
2021-09-23 03:51:14.829 INFO    numexpr.utils: NumExpr defaulting to 8 threads.

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.2:8501

2021-09-23 03:51:20.062 Initializing load_model()
2021-09-23 03:51:20.062 load_model(model_name=classification_model, platform=No\
ne, authentication=None, verbose=True)
Transformation Pipeline and Model Successfully Loaded
```

Deploying an Application to Streamlit Cloud

We have now developed two Streamlit applications based on PyCaret models, so the next step is to deploy them on Streamlit Cloud. In the rest of this chapter, we are going to walk through the steps of deploying the Insurance Charges Prediction App on the Community tier of Streamlit Cloud. Given that the process is practically identical for every application, we are skipping the Iris Classification App, but you are welcome to deploy it yourself!

Creating a Github Repository

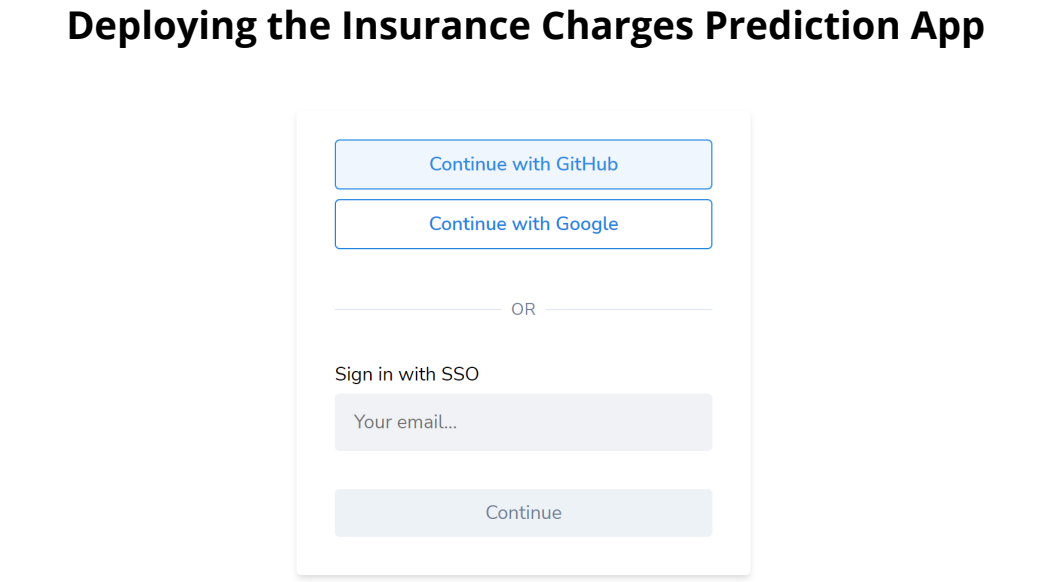


We begin by creating a Github repository for the application, including the source code, as well as the regression model and `requirements.txt`, which is the dependencies file. In this case, `requirements.txt` contains the following dependency only, which specifies the PyCaret version to be installed.

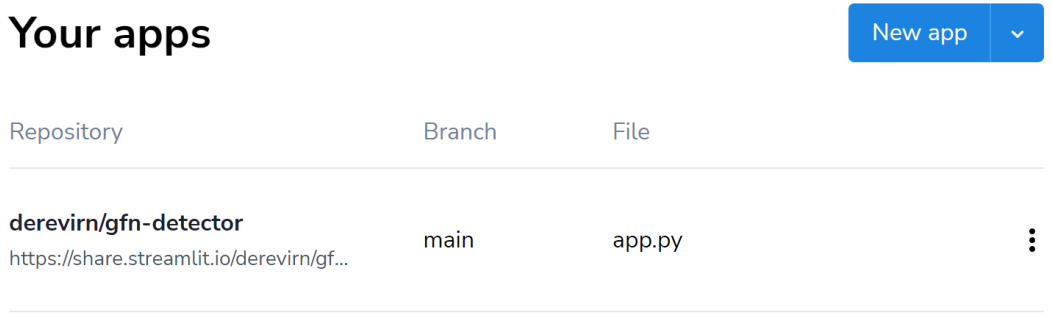
```
pycaret==2.3.4
```

In case you want to fork or clone the Insurance Charges Prediction App repository, it is available at this link[7] . The complete code of this chapter is available at the official book repository[8] as well.

[7] https://github.com/derevirn/pycaret-insurance
[8] https://github.com/derevirn/pycaret-book

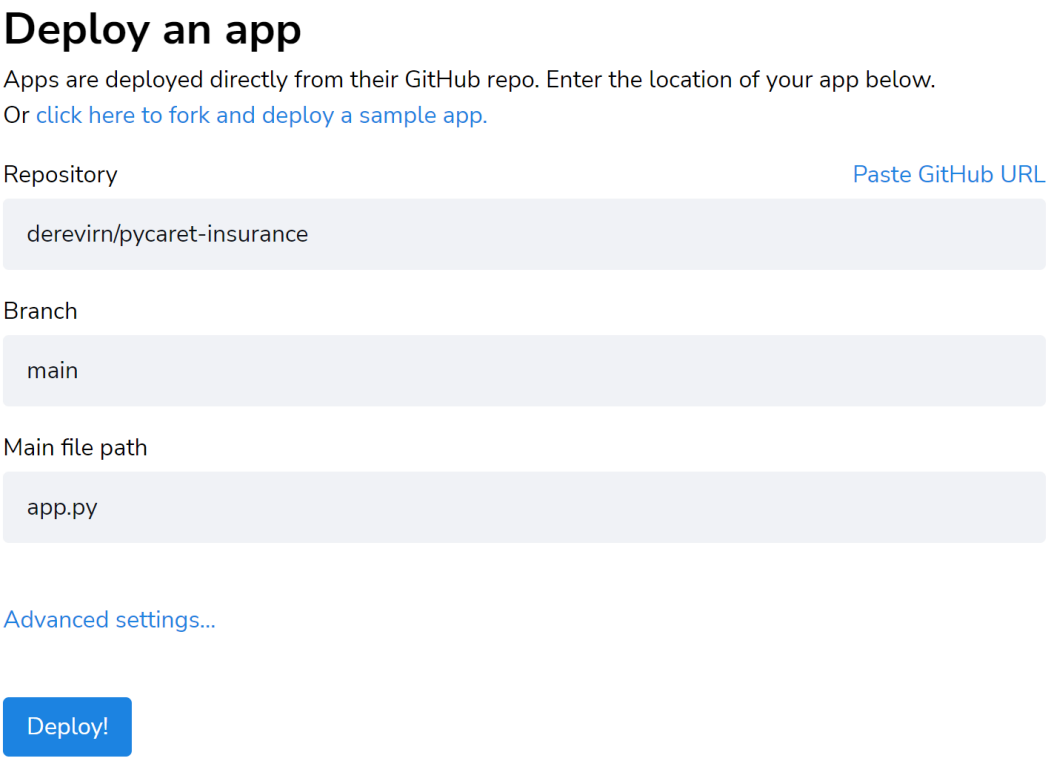


As mentioned before, the Community tier of Streamlit Cloud is freely available, so you can visit this link[9] and sign in to the service by using your Github account. Alternatively, you can also use your Google account or the Single Sign-on (SSO) option if you prefer.

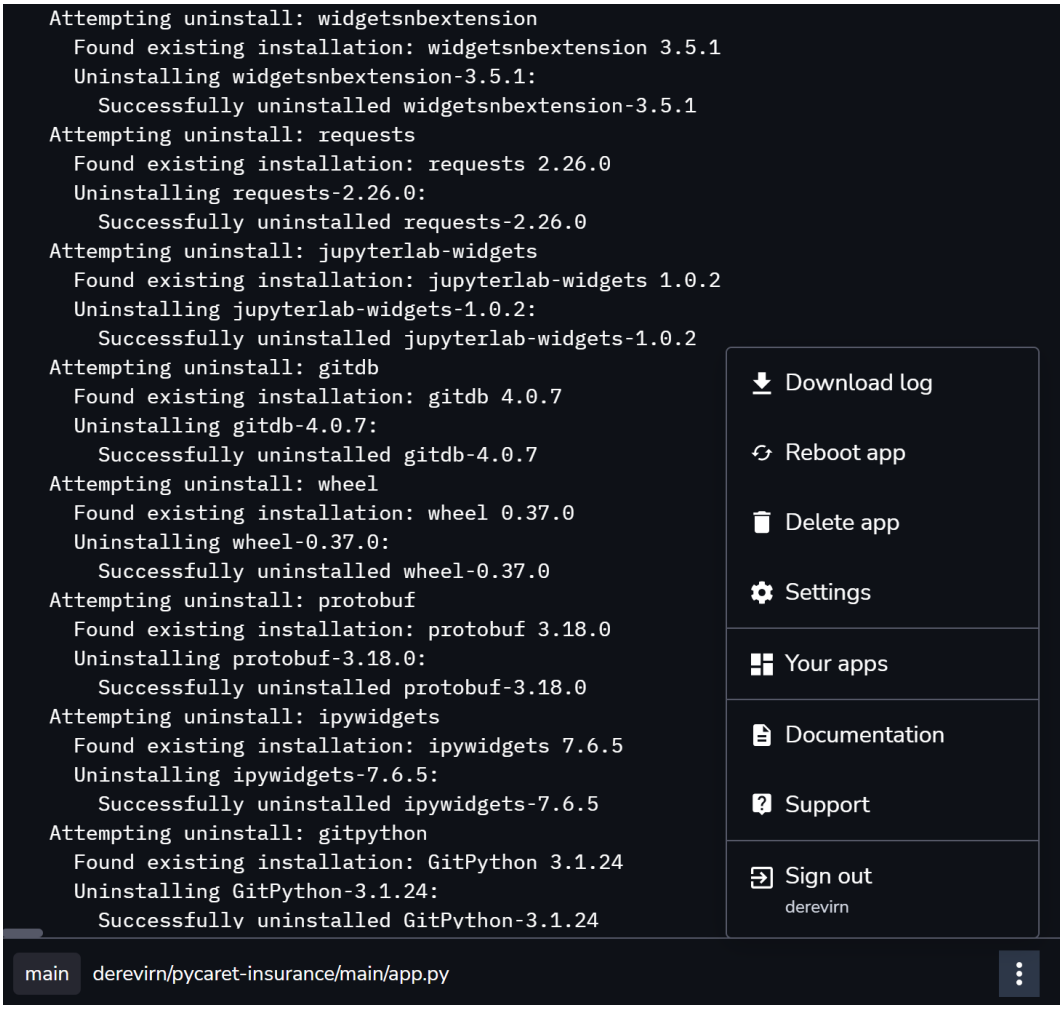


After signing in, you can deploy an application simply by clicking the **New app** button. You can also see a list of the deployed apps, or change their settings by clicking the menu icon. Furthermore, you can also change the workspace settings or sign out of Streamlit cloud. As you can see, I have already deployed another Streamlit application, that is based on a personal project about fake news detection.

[9] https://share.streamlit.io/



After clicking the **New app** button, you have to enter some necessary details about the application that is going to be deployed. First of all, you must select one of the repositories that are publicly available on your Github account. Second, you have to pick the repository branch, with main being the default option. Finally, you have to choose the path of the main application file, typically named `app.py`. I encourage you to put this file in the root folder of the repository, as you may encounter some errors if you do otherwise. The advanced settings option gives you the ability to change the Python version, as well as set environment variables, but you can safely ignore those settings for now. After entering the application details, simply click the **Deploy** button, and wait until the process is complete.



Anybody can use the application now, simply by visiting the provided URL. For example, you can check the Insurance Charges Prediction App by clicking here[10]. An extra button named **Manage app** is added at the bottom right corner of the application, that is only visible to us. By clicking that, you can view the console output of the Streamlit Cloud server. This can help you check any potential errors, or see the current status of the application. Furthermore, by clicking the menu icon, you can reboot or delete the application. Other options include changing the settings, downloading the server logs and signing out.

[10] https://share.streamlit.io/derevirn/pycaret-insurance/main/app.py

8. Closing Thoughts

In this book, we covered multiple topics related to machine learning. We started with a brief introduction to the basic theoretical concepts, and then continued with case studies of regression, classification, clustering, anomaly detection and natural language processing, based on the respective modules of the PyCaret library. After doing that, we focused on using the Streamlit library to develop and deploy machine learning applications. It should be noted that machine learning is a dynamic and ever-evolving field, so we have only scratched the surface of everything there is to know about it.

Regardless, now that you have finished reading this book, you should have a fairly solid understanding of the fundamental machine learning tasks and techniques, as well as the PyCaret library features. Having said that, I encourage you to consult the PyCaret documentation[1] so you can learn about the modules and functions that haven't been covered in the book (yet). I also suggest that you keep experimenting with datasets that interest you, and create a portfolio of projects that showcase your abilities and experience.

Feel free to contact me if you have any issues or problems with the book, as I will be glad to help you resolve them. I also encourage you to share your thoughts and feedback by filling in this form[2]. Writing this book was a challenging process that required numerous hours of hard work. Nevertheless, I am truly proud of the result, and I sincerely hope that you gain value from reading it. Keep in mind that education is an everlasting process, so this book can't teach you everything, but I hope it motivates you to continue learning and growing as a professional!

[1] https://pycaret.readthedocs.io/en/latest/
[2] https://forms.gle/1hsbBtG1ZSEcyrH27

Notes

A Brief Introduction to Machine Learning

1 Samuel, Arthur L. "Some studies in machine learning using the game of checkers." IBM Journal of research and development 3.3 (1959): 210-229.

2 Raschka, Sebastian. Python machine learning. Packt publishing ltd, 2015.

3 Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

4 Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

5 Kiran, B. Ravi, et al. "Deep reinforcement learning for autonomous driving: A survey." IEEE Transactions on Intelligent Transportation Systems (2021).

Regression

6 Olive, David J. "Multiple linear regression." Linear regression. Springer, Cham, 2017. 17-83.

7 Tibshirani, Robert. "Regression shrinkage and selection via the lasso." Journal of the Royal Statistical Society: Series B (Methodological) 58.1 (1996): 267-288.

8 Ho, Tin Kam. "Random decision forests." Proceedings of 3rd international conference on document analysis and recognition. Vol. 1. IEEE, 1995.

9 Drucker, Harris, et al. "Support vector regression machines." Advances in neural information processing systems 9 (1997): 155-161.

10 Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

11 Lantz, Brett. Machine learning with R: expert techniques for predictive modeling. Packt publishing ltd, 2019.

Classification

12 Kleinbaum, David G., et al. Logistic regression. New York: Springer-Verlag, 2002.

13 Navada, Arundhati, et al. "Overview of use of decision tree algorithms in machine learning." 2011 IEEE control and system graduate research colloquium. IEEE, 2011.

14 Mucherino, Antonio, Petraq J. Papajorgji, and Panos M. Pardalos. "K-nearest neighbor classification." Data mining in agriculture. Springer, New York, NY, 2009. 83-106.

15 Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." the Journal of machine Learning research 3 (2003): 993-1022.

16 Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.

17 Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." Advances in neural information processing systems 25 (2012).

Clustering

18 Steinley, Douglas. "K-means clustering: a half-century synthesis." British Journal of Mathematical and Statistical Psychology 59.1 (2006): 1-34.

19 Dueck, Delbert. Affinity propagation: clustering data by passing messages. Toronto: University of Toronto, 2009.

20 Von Luxburg, Ulrike. "A tutorial on spectral clustering." Statistics and computing 17.4 (2007): 395-416.

21 Ackermann, Marcel R., et al. "Analysis of agglomerative clustering." Algorith-

mica 69.1 (2014): 184-215.

22 Derpanis, Konstantinos G. "Mean shift clustering." Lecture Notes (2005): 32.

23 Khan, Kamran, et al. "DBSCAN: Past, present and future." The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014). IEEE, 2014.

24 Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of computational and applied mathematics 20 (1987): 53-65.

25 Caliński, Tadeusz, and Jerzy Harabasz. "A dendrite method for cluster analysis." Communications in Statistics-theory and Methods 3.1 (1974): 1-27.

26 Davies, David L., and Donald W. Bouldin. "A cluster separation measure." IEEE transactions on pattern analysis and machine intelligence 2 (1979): 224-227.

27 Yuan, Chunhui, and Haitao Yang. "Research on K-value selection method of K-means clustering algorithm." J 2.2 (2019): 226-235.

28 Abdi, Hervé, and Lynne J. Williams. "Principal component analysis." Wiley interdisciplinary reviews: computational statistics 2.4 (2010): 433-459.

Anomaly Detection

29 Breunig, Markus M., et al. "LOF: identifying density-based local outliers." Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000.

30 Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." 2008 eighth ieee international conference on data mining. IEEE, 2008.

31 Ramaswamy, Sridhar, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets." Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000.

32 Kriegel, Hans-Peter, et al. "Outlier detection in axis-parallel subspaces of high

dimensional data." Pacific-asia conference on knowledge discovery and data mining. Springer, Berlin, Heidelberg, 2009.

33 He, Zengyou, Xiaofei Xu, and Shengchun Deng. "Discovering cluster-based local outliers." Pattern Recognition Letters 24.9-10 (2003): 1641-1650.

34 McInnes, Leland, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction." arXiv preprint arXiv:1802.03426 (2018).

35 Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.11 (2008).

Natural Language Processing

36 Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." the Journal of machine Learning research 3 (2003): 993-1022.

37 Hofmann, Thomas. "Probabilistic latent semantic indexing." Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. 1999.

38 Teh, Yee Whye, et al. "Hierarchical dirichlet processes." Journal of the american statistical association 101.476 (2006): 1566-1581.

39 Kanerva, Pentii, Jan Kristoferson, and Anders Holst. "Random indexing of text samples for latent semantic analysis." Proceedings of the Annual Meeting of the Cognitive Science Society. Vol. 22. No. 22. 2000.

40 Zhao, Renbo, and Vincent YF Tan. "Online nonnegative matrix factorization with outliers." IEEE Transactions on Signal Processing 65.3 (2016): 555-570.

41 Sievert, Carson, and Kenneth Shirley. "LDAvis: A method for visualizing and interpreting topics." Proceedings of the workshop on interactive language learning, visualization, and interfaces. 2014.