

The Data Science





Table of Contents

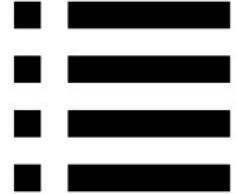


1. Introduction to Data Science in Python: 4
2. Regression: 67
3. Binary Classification: 97
4. Multiclass Classification with RandomForest : 159
5. Performing Your First Cluster Analysis: 199
6. How to Assess Performance: 257
7. The Generalization of Machine Learning Models: 305
8. Hyperparameter Tuning: 347
9. Interpreting a Machine Learning Model: 424





Table of Contents



10. Analyzing a Dataset: 471
11. Data Preparation: 532
12. Feature Engineering: 581
13. Imbalanced Datasets: 643
14. Dimensionality Reduction: 673
15. Ensemble Learning: 709



1. Introduction to Data Science in Python



Overview

- This very first lesson will introduce you to the field of data science and walk you through an overview of Python's core concepts and their application in the world of data science.



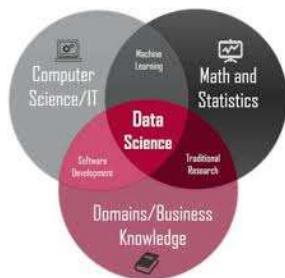
Introduction

- Welcome to the fascinating world of data science! We are sure you must be pretty excited to start your journey and learn interesting and exciting techniques and algorithms, This is exactly what this course is intended for.



Application of Data Science

- Defining the business problem to be solved
- Collecting or extracting existing data
- Analyzing, visualizing, and preparing data
- Training a model to spot patterns in data and make predictions
- Assessing a model's performance and making improvements
- Communicating and presenting findings and gained insights
- Deploying and maintaining a model



What Is Machine Learning?

Machine learning is composed of three different types of learning:

- Supervised learning
- Unsupervised learning
- Reinforcement learning



Supervised Learning

- Supervised learning refers to a type of task where an algorithm is trained to learn patterns based on prior knowledge.
- That means this kind of learning requires the labeling of the outcome (also called the response variable, dependent variable, or target variable) to be predicted beforehand.

Target

Features



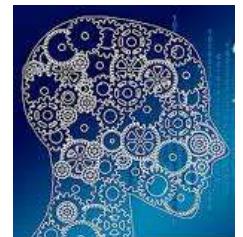
Cancel	Months since first subscription	Monthly average spent	Average number of phone calls made last month	Average number of phone calls made last quarter	Additional options
Yes	13	\$70	56	63	Yes
No	2	\$35	35	34	Yes
No	6	\$40	46	50	Yes
Yes	16	\$110	53	75	No

Supervised Learning

Daily output	Plant ID	Number of staff	Stock of screens	Stock of 5G chip
53003	A	124	102432	0
21342	N	54	30132	0
42032	C	103	125312	0
15234	E	84	42232	50234

Unsupervised Learning

- Unsupervised learning is a type of algorithm that doesn't require any response variables at all.
- In this case, the model will learn patterns from the data by itself.
- You may ask what kind of pattern it can find if there is no target specified beforehand.



Reinforcement Learning

- Reinforcement learning is another type of algorithm that learns how to act in a specific environment based on the feedback it receives.
- You may have seen some videos where algorithms are trained to play Atari games by themselves.
- Reinforcement learning techniques are being used to teach the agent how to act in the game based on the rewards or penalties it receives from the game.

Overview of Python

Numeric Variables

- The most basic variable type is numeric.
- This can contain integer or decimal (or float) numbers, and some mathematical operations can be performed on top of them.



Numeric Variables

- Let's use an integer variable called var1 that will take the value 8 and another one called var2 with the value 160.88, and add them together with the + operator, as shown here:

var1 = 8

var2 = 160.88

var1 + var2

- You should get the following output:

168.88



Text Variables

- Another interesting type of variable is string, which contains textual information.
- You can create a variable with some specific text using the single or double quote, as shown in the following example:

```
var3 = 'Hello, '
var4 = 'World'
```



Text Variables

- In order to display the content of a variable, you can call the `print()` function:

`print(var3)`

`print(var4)`

- You should get the following output:

Hello,
World



Text Variables

- For instance, if we want to print Text: before the values of var3 and var4, we will write the following code:

```
print(f"Text: {var3} {var4}!")
```

- You should get the following output:

Text: Hello, World!



Text Variables

- You can also perform some text-related transformations with string variables, such as capitalizing or replacing characters.
- For instance, you can concatenate the two variables together with the + operator:

`var3 + var4`

- You should get the following output:

`'Hello, World'`



Python List

- Another very useful type of variable is the list.
- It is a collection of items that can be changed (you can add, update, or remove items).
- To declare a list, you will need to use square brackets, [], like this:

```
var5 = ['I', 'love', 'data', 'science']
print(var5)
```

- You should get the following output:

```
['I', 'love', 'data', 'science']
```



Python List

- A list can have different item types, so you can mix numerical and text variables in it:

```
var6 = ['Ernesto', 15019, 2020, 'Data Science']
print(var6)
```

- You should get the following output:

```
[ 'Ernesto', 15019, 2020, 'Data Science' ]
```



Python List

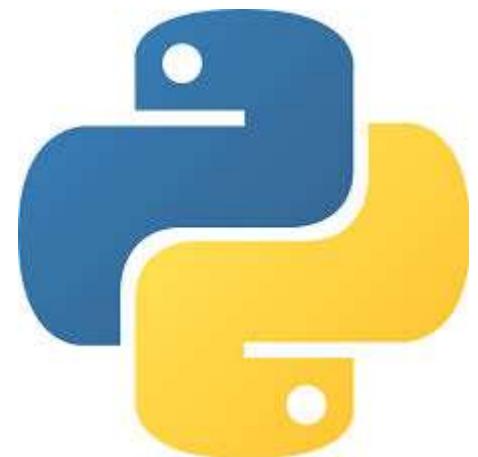
- An item in a list can be accessed by its index (its position in the list).
- To access the first (index 0) and third elements (index 2) of a list, you do the following:

```
print(var6[0])  
print(var6[2])
```

Python List

- You should get the following output:

Ernesto
2020



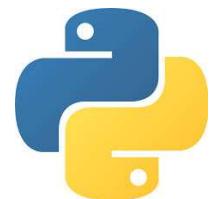
Python List

- You can also iterate through every item of a list using a for loop.
- If you want to print every item of the var6 list, you should do this:

```
for item in var6:  
    print(item)
```

- You should get the following output:

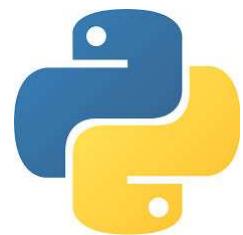
Ernesto
15019
2020
Data Science



Python List

- You can add an item at the end of the list using the `.append()` method:

```
var6.append('Python')  
print(var6)
```



- You should get the following output:

```
[ 'Ernesto', 15019, 2020, 'Data Science', 'Python' ]
```

Python List

- To delete an item from the list, you use the `.remove()` method:

```
var6.remove(15019)  
print(var6)
```



- You should get the following output:

```
['Ernesto', 2020, 'Data Science', 'Python']
```

Python Dictionary

- To define a dictionary in Python, you will use curly brackets, {}, and specify the keys and values separated by :, as shown here:

```
var7 = {'Topic': 'Data Science', 'Language': 'Python'}  
print(var7)
```

- You should get the following output:

```
{ 'Topic' : 'Data Science', 'Language' : 'Python' }
```

Python Dictionary

- To access a specific value, you need to provide the corresponding key name.
- For instance, if you want to get the value Python, you do this:

```
var7['Language']
```

- You should get the following output:

```
'Python'
```



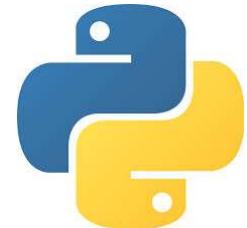
Python Dictionary

- Python provides a method to access all the key names from a dictionary, .keys(), which is used as shown in the following code snippet:

`var7.keys()`

- You should get the following output:

```
dict_keys(['Topic', 'Language'])
```



Python Dictionary

- There is also a method called `.values()`, which is used to access all the values of a dictionary:

`var7.values()`

- You should get the following output:

```
dict_values(['Data Science', 'Python'])
```



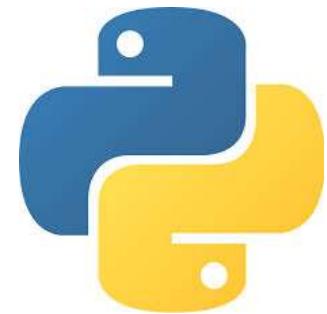
Python Dictionary

- You can iterate through all items from a dictionary using a for loop and the .items() method, as shown in the following code snippet:

```
for key, value in var7.items():
    print(key)
    print(value)
```

- You should get the following output:

Topic
Data Science
Language
Python



Python Dictionary

- You can add a new element in a dictionary by providing the key name like this:

```
var7['Publisher'] = 'Ernesto'  
print(var7)
```

- You should get the following output:

```
{'Topic': 'Data Science', 'Language': 'Python', 'Publisher': 'Ernesto'}
```

Python Dictionary

- You can delete an item from a dictionary with the `del` command:

```
del var7['Publisher']
print(var7)
```

- You should get the following output:

```
{'Topic': 'Data Science', 'Language': 'Python'}
```

Complete Exercise : Creating a Dictionary That Will Contain Machine Learning Algorithms

Python for Data Science

The pandas Package

- The pandas package provides an incredible amount of APIs for manipulating data structures.
- The two main data structures defined in the pandas package are DataFrame and Series.



DataFrame and Series

- A DataFrame is a tabular data structure that is represented as a two-dimensional table.
- It is composed of rows, columns, indexes, and cells.
- It is very similar to a sheet in Excel or a table in a database.

The diagram illustrates a 4x3 data table with the following structure:

	algorithm	learning	type
0	Linear Regression	Supervised	Regression
1	Logistic Regression	Supervised	Classification
2	RandomForest	Supervised	Regression or Classification
3	k-means	Unsupervised	Clustering

Annotations highlight specific parts of the table:

- Column:** An orange bracket on the left indicates the column for the "algorithm" header.
- Index:** A blue bracket on the left indicates the index for the first three rows (0, 1, 2).
- Row:** A green bracket on the right indicates the row for the "type" header.
- Cell:** A red bracket on the right indicates the cell for "k-means".

CSV Files

- CSV files use the comma character—,—to separate columns and newlines for a new row.
- The previous example of a DataFrame would look like this in a CSV file:

algorithm,learning,type

Linear Regression,Supervised,Regression

Logistic Regression,Supervised,Classification

RandomForest,Supervised,Regression or Classification

k-means,Unsupervised,Clustering

CSV Files

- In Python, you need to first import the packages you require before being able to use them.
- To do so, you will have to use the import command.
- You can create an alias of each imported package using the as keyword.
- It is quite common to import the pandas package with the alias pd:

```
import pandas as pd
```

CSV Files

```
pd.read_csv('https://raw.githubusercontent.com/fenago'\
    '/data-science/master/Lab01/'\
    'Dataset/csv_example.csv')
```



CSV Files

	algorithm	learning	type
0	Linear Regression	Supervised	Regression
1	Logistic Regression	Supervised	Classification
2	RandomForest	Supervised	Regression or Classification
3	k-means	Unsupervised	Clustering

Excel Spreadsheets

- Excel is a Microsoft tool and is very popular in the industry.
- It has its own internal structure for recording additional information, such as the data type of each cell or even Excel formulas.
- There is a specific method in pandas to load Excel spreadsheets called `.read_excel()`:

```
pd.read_excel('https://github.com/fenago'\
              '/data-science/blob/master'\
              '/Lab01/Dataset/excel_example.xlsx?raw=true')
```

Excel Spreadsheets

	algorithm	learning	type
0	Linear Regression	Supervised	Regression
1	Logistic Regression	Supervised	Classification
2	RandomForest	Supervised	Regression or Classification
3	k-means	Unsupervised	Clustering

JSON

- JSON is a very popular file format, mainly used for transferring data from web APIs.
- Its structure is very similar to that of a Python dictionary with key-value pairs.
- The example DataFrame we used before would look like this in JSON format:

Refer to the file 1_1.txt



JSON

- As you may have guessed, there is a pandas method for reading JSON data as well, and it is called `.read_json()`:

```
pd.read_json('https://raw.githubusercontent.com/fenago'  
\\  
'/data-science/master/Lab01'\\  
'/Dataset/json_example.json')
```

JSON

	algorithm	learning	type
0	Linear Regression	Supervised	Regression
1	Logistic Regression	Supervised	Classification
2	RandomForest	Supervised	Regression or Classification
3	k-means	Unsupervised	Clustering

Complete Exercise : Loading Data of Different Formats into a pandas DataFrame

Scikit-Learn

What Is a Model?

- A machine learning model learns patterns from data and creates a mathematical function to generate predictions.
- A supervised learning algorithm will try to find the relationship between a response variable and the given features.
- Have a look at the following example.

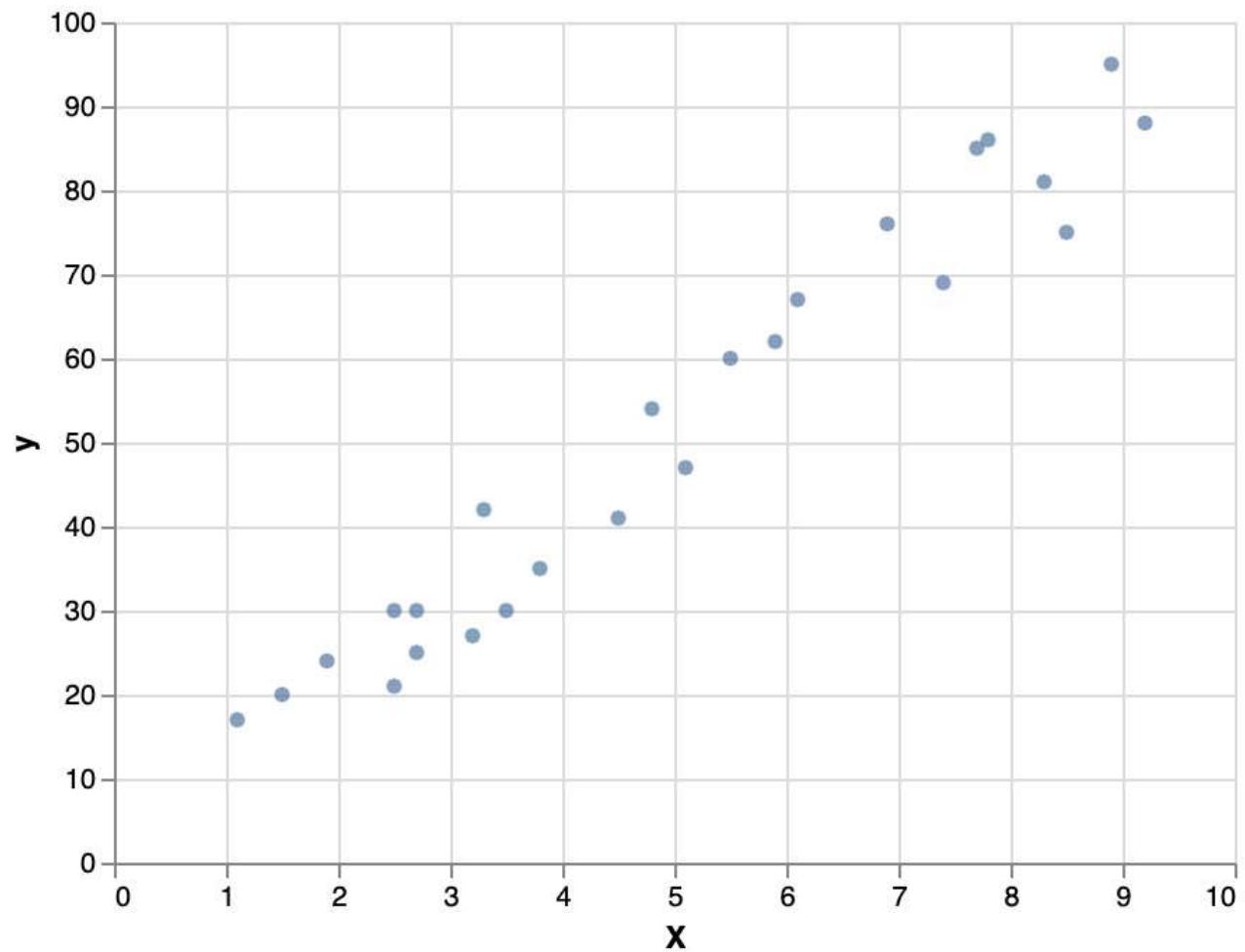
$$\hat{y} = f(X) = f(x_1, x_2, \dots, x_n)$$



Scikit-Learn

- The function, $f()$, can be quite complex and have different numbers of parameters.
- If we take a linear regression (this will be presented in more detail in lesson 2, Regression) as an example, the model parameters can be represented as $W=(w_1, w_2, \dots, w_n)$. So, the function we saw earlier will become as follows:

$$f(X) = f(x_1, x_2, \dots, x_n) = w_0 + w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n = \hat{y}$$



Scikit-Learn

- If we fit a linear regression on this dataset, the algorithm will try to find a solution for the following equation:

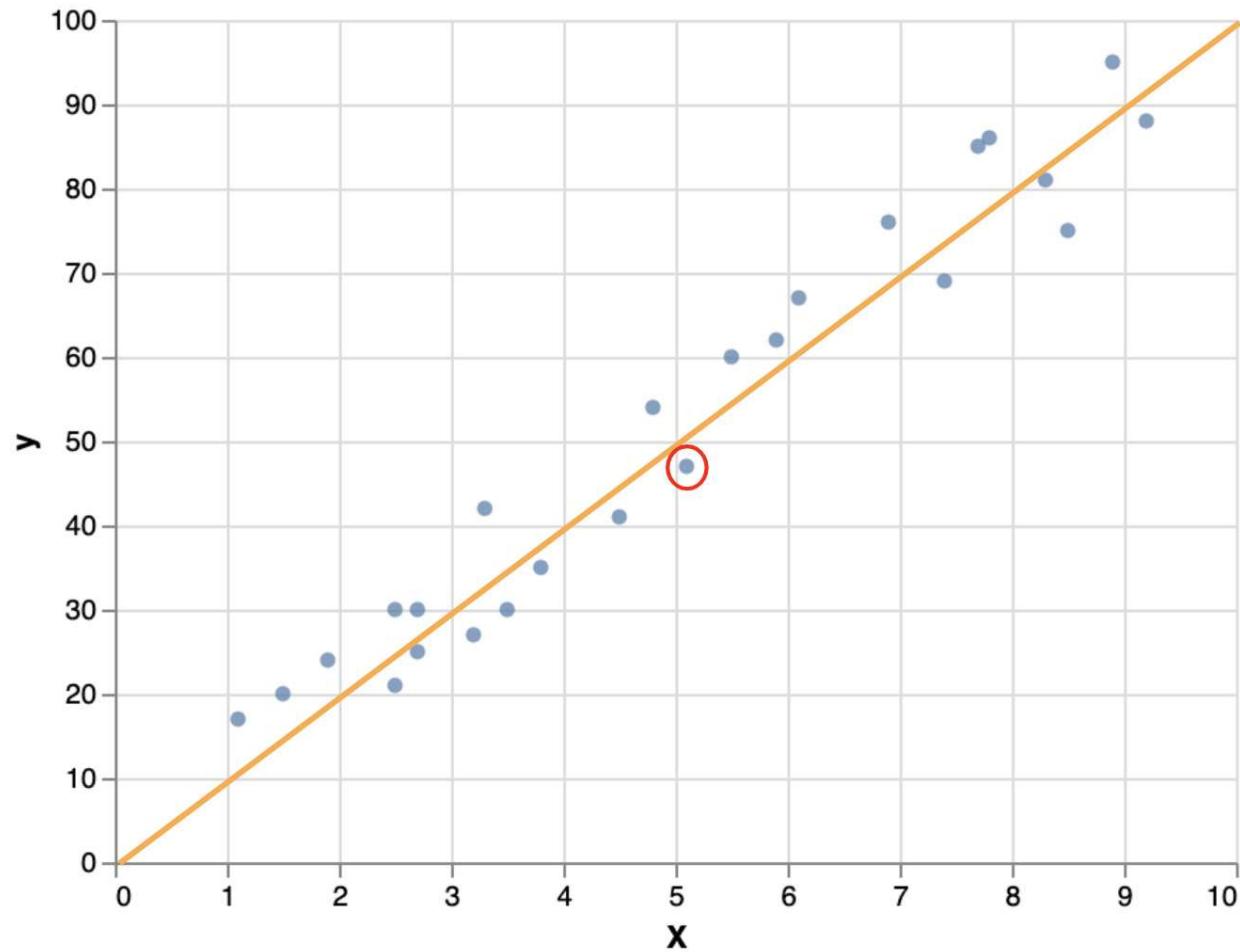
$$f(X) = w_0 + w_1 * x_1$$



Scikit-Learn

- o, it just needs to find the values of the w_0 and w_1 parameters that will approximate the data as closely as possible.
- In this case, the algorithm may come up with $w_0 = 0$ and $w_1 = 10$. So, the function the model learns will be as follows:

$$f(X) = 0 + 10 * x_1 = 10 * x_1$$



Scikit-Learn

- We can see that the fitted model (the orange line) is approximating the original data quite closely.
- So, if we predict the outcome for a new data point, it will be very close to the true value.
- For example, if we take a point that is close to 5 (let's say its values are $x = 5.1$ and $y = 48$), the model will predict the following:

$$\hat{y} = 10 * x = 10 * 5.1 = 51$$

Model Hyperparameters

- On top of the model parameters that are learned automatically by the algorithm (now you understand why we call it machine learning), there is also another type of parameter called the hyperparameter.
- Hyperparameters cannot be learned by the model.
- They are set by data scientists in order to define some specific conditions for the algorithm learning process.

The sklearn API

- sklearn groups algorithms by family. For instance, RandomForest and GradientBoosting are part of the ensemble module.
- In order to make use of an algorithm, you will need to import it first like this:

```
from sklearn.ensemble import RandomForestClassifier
```

The sklearn API

- It is recommended to at least set the `random_state` hyperparameter in order to get reproducible results every time that you have to run the same code:

```
rf_model = RandomForestClassifier(random_state=1)
```



The sklearn API

- The second step is to train the model with some data.
- In this example, we will use a simple dataset that classifies 178 instances of Italian wines into 3 categories based on 13 features.
- This dataset is part of the few examples that sklearn provides within its API. We need to load the data first:

```
from sklearn.datasets import load_wine  
features, target = load_wine(return_X_y=True)
```

The sklearn API

- Then using the `.fit()` method to train the model, you will provide the features and the target variable as input:

`rf_model.fit(features, target)`

- You should get the following output:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=1, verbose=0,
                      warm_start=False)
```

The sklearn API

- Once trained, we can use the .predict() method to predict the target for one or more observations.
 - Here we will use the same data as for the training step:
`preds = rf_model.predict(features)`
`preds`
 - You should get the following output:

The sklearn API

- For now, though, we will just use a metric called accuracy.
- This metric calculates the ratio of correct predictions to the total number of observations:
`from sklearn.metrics import accuracy_score
accuracy_score(target, preds)`

- You should get the following output

1.0



Complete Exercise : Predicting Breast Cancer from a Dataset Using sklearn

Complete Activity : Train a Spam Detector Algorithm

Summary

- This lesson provided you with an overview of what data science is in general.
- We also learned the different types of machine learning algorithms, including supervised and unsupervised, as well as regression and classification.
- We had a quick introduction to Python and how to manipulate the main data structures (lists and dictionaries) that will be used in this course.



2. Regression

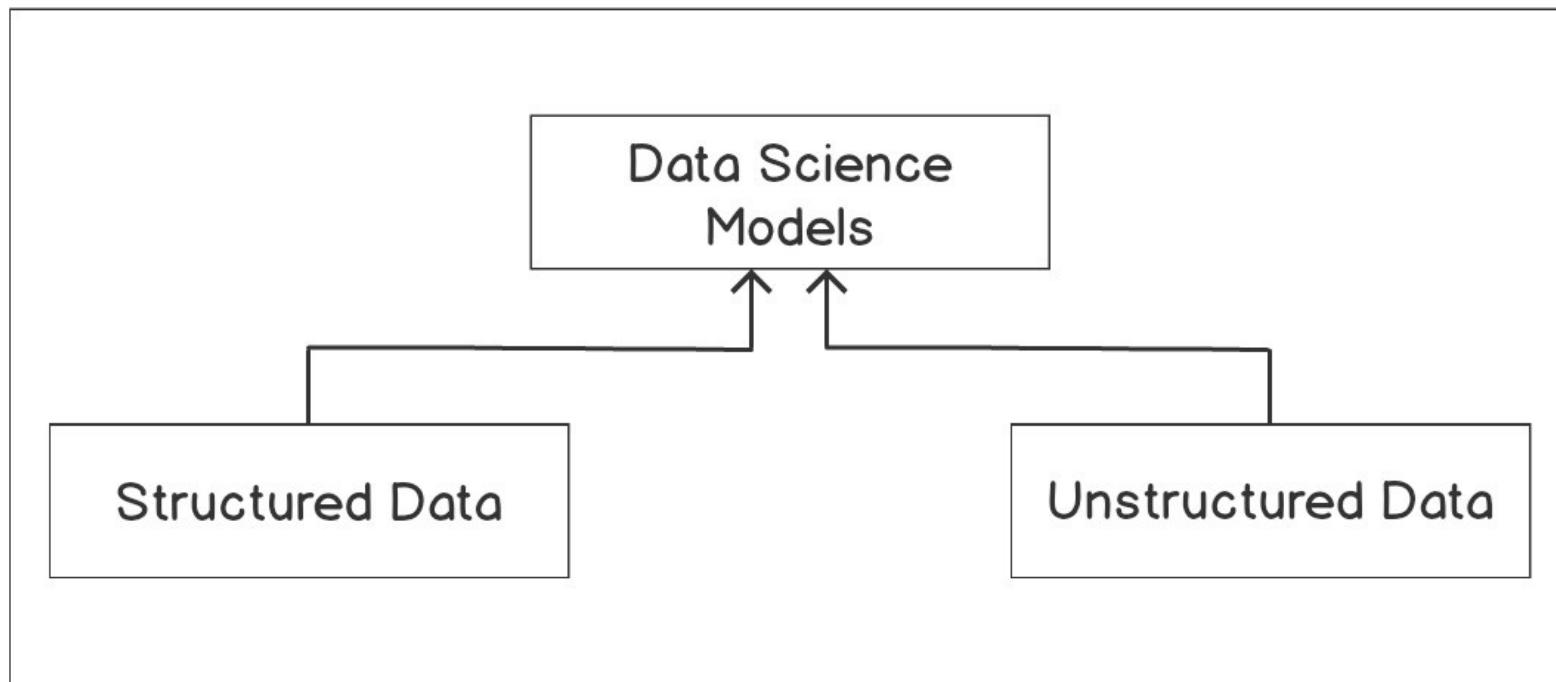


Overview

- This lesson is an introduction to linear regression analysis and its application to practical problem-solving in data science.
 - You will learn how to use Python, a versatile programming language, to carry out regression analysis and examine the results.



Introduction



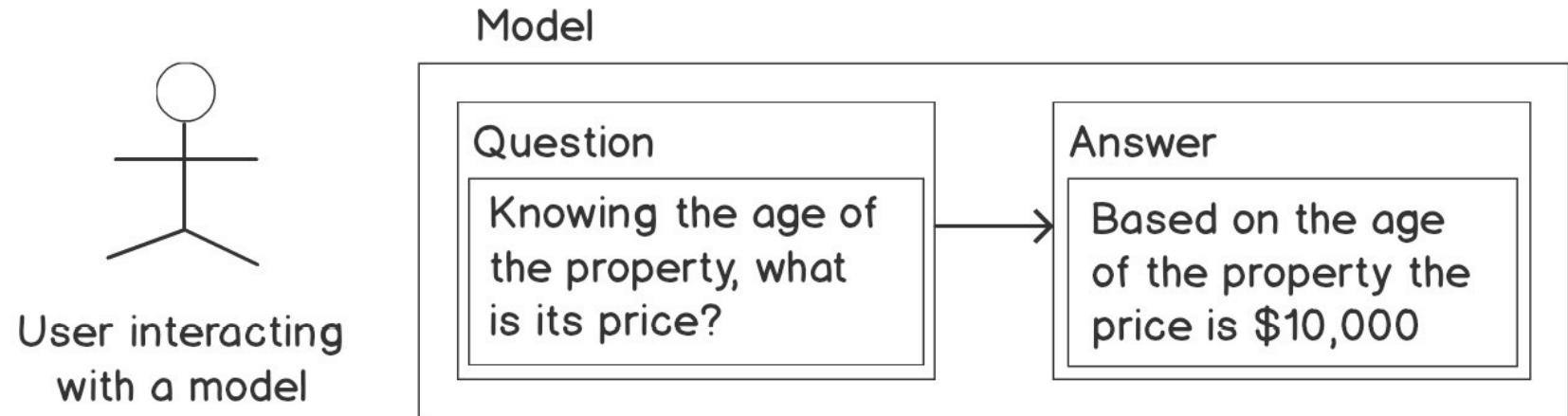
Introduction

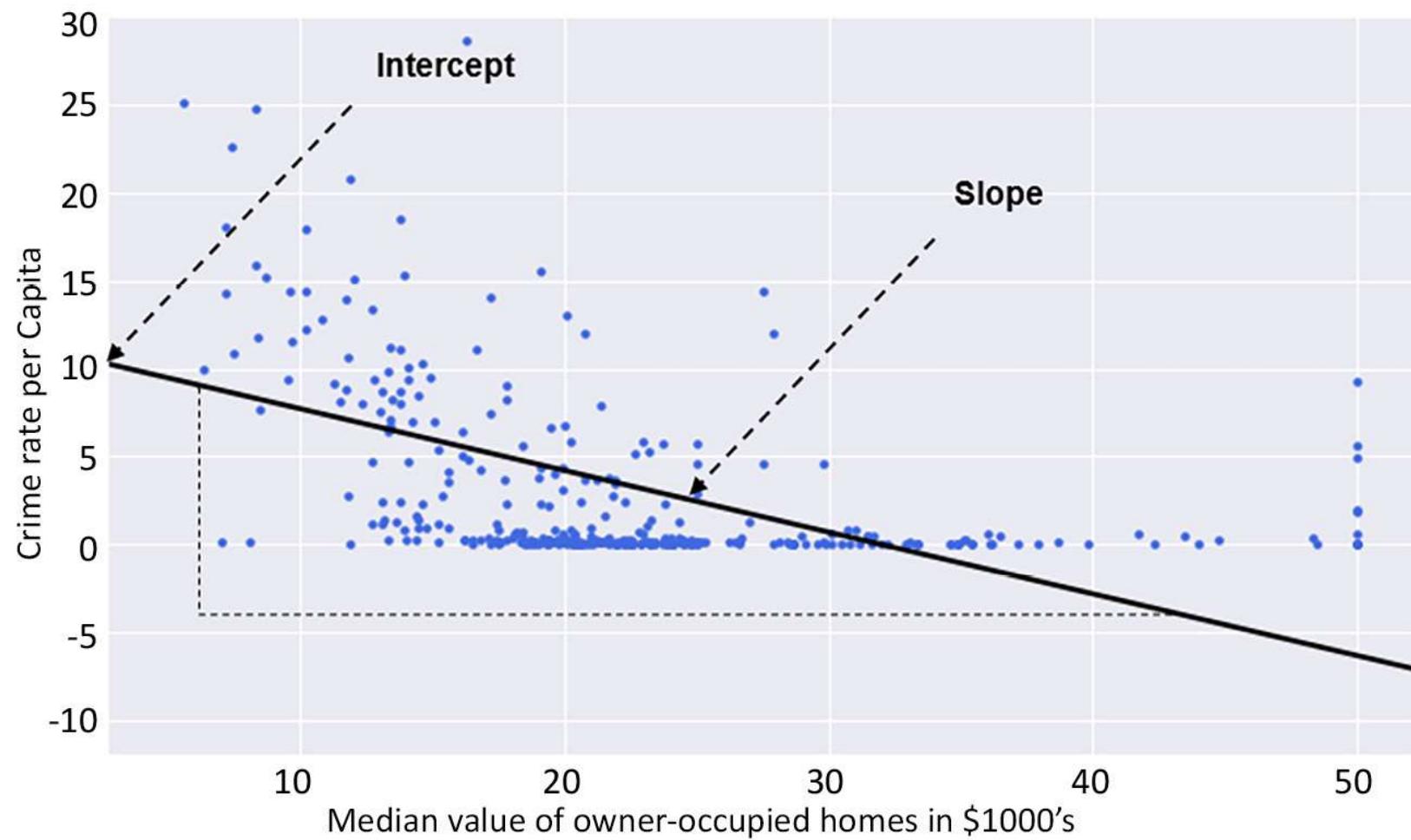
Examples of the key attributes of a property that can be used to predict its value are as follows:

- The age of the property
- The number of bedrooms in a property
- Whether the property has a pool or not
- The area of land the property covers
- The distance of the property from facilities such as railway stations and schools

Introduction

- Regression analysis can be employed to study this scenario, in which you have to create a function that maps the key attributes of a property to the target variable, which, in this case, is the price of a property.





The Method of Least Squares

- The simple linear regression line is generally of the form shown in next Figure, where β_0 and β_1 are unknown constants, representing the intercept and the slope of the regression line, respectively.
- The intercept is the value of the dependent variable (Y) when the independent variable (X) has a value of zero (0).

$$Y \approx \beta_0 + \beta_1 X$$

Multiple Linear Regression

- Let's consider a case where we want to fit a linear regression model that has three independent variables, X_1 , X_2 , and X_3 .
- The formula for the multiple linear regression equation will look like

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$$

Conducting Regression Analysis Using Python

- Having discussed the basics of regression analysis, it is now time to get our hands dirty and actually do some regression analysis using Python.
- To begin with our analysis, we need to start a session in Python and load the relevant modules and dataset required.
- All of the regression analysis we will do in this lesson will be based on the Boston Housing dataset.
- The dataset is good for teaching and is suitable for linear regression analysis.

Complete Exercise 2.01: Loading and Preparing the Data for Analysis

The Correlation Coefficient

- In the previous exercise, we have seen how a correlation matrix heatmap can be used to visualize the relationships between pairs of variables.
- We can also see these same relationships in numerical form using the raw correlation coefficient numbers.
- These are values between -1 and 1, which represent how closely two variables are linked.

The Correlation Coefficient

- You should get the following output:

	landOver25K_sqft	non-retailLandProptn	riverDummy	nitrixOxide_pp10m	avgNo.RoomsPerDwelling
landOver25K_sqft	1.000000	-0.540095	-0.059189	-0.520305	0.355346
non-retailLandProptn	-0.540095	1.000000	0.065271	0.758178	-0.399166
riverDummy	-0.059189	0.065271	1.000000	0.091469	0.107996
nitrixOxide_pp10m	-0.520305	0.758178	0.091469	1.000000	-0.306510
avgNo.RoomsPerDwelling	0.355346	-0.399166	0.107996	-0.306510	1.000000
proptnOwnerOccupied	-0.577457	0.667887	0.106329	0.742016	-0.263085
weightedDist	0.659340	-0.728968	-0.098551	-0.776311	0.215439
radialHighwaysAccess	-0.311920	0.580813	0.022731	0.606721	-0.183000
propTaxRate_per10K	-0.324172	0.702973	-0.007864	0.662164	-0.280341

Complete Exercise 2.02: Graphical Investigation of Linear Relationships Using Python

Complete Exercise 2.03: Examining a Possible Log-Linear Relationship Using Python

The Statsmodels formula API

- A solid line represents the relationship between the crime rate per capita and the median value of owner-occupied homes.
- But how can we obtain the equation that describes this line?
- In other words, how can we find the intercept and the slope of the straight-line relationship?

Complete Exercise 2.04: Fitting a Simple Linear Regression Model Using the Statsmodels formula API

Analyzing the Model Summary

- The `.fit` method provides many functions to explore its output.
- These include the `conf_int()`, `pvalues`, `tvalues`, and `summary()` parameters.
- With these functions, the parameters of the model, the confidence intervals, and the p-values and t-values for the analysis can be retrieved from the results. (The concept of p-values and t-values will be explained later in the lesson.)

The Model Formula Language

Operator Symbol	Meaning	Example
\sim	Separates the left-hand side and the right-hand side of a formula.	$Y \sim X$ will define the model $Y = \beta_0 + \beta X$.
$+$	This is used to include an independent variable in the model (not arithmetic addition). It computes a set union of terms given on its left and those given on its right.	$Y \sim X_1 + X_2$ will define the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$.
$-$	This is used to delete an independent variable from the model (not arithmetic subtraction). It computes a set difference by removing terms given on its right from terms given on its left.	$Y \sim X - 1$ will define the model $Y = \beta X$. The intercept is defined as 1, hence the minus operator (-) deletes it from this model. This model, therefore, goes through the origin. See more on intercept handling after this table.
$*$	This is used to include independent variables on its left and right and their interactions in a model (not arithmetic multiplication).	$Y \sim X_1 * X_2$ will define the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$.

/	This indicates nesting of independent variables in the model (not arithmetic division). Intended to be useful in cases where one wants to fit a standard analysis of variance (ANOVA) model but one term is nested in another term.	$Y \sim X_1 / X_2$ will define the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1 X_2$. Here, X_2 is a term nested within X_1 .
:	The colon operator includes a pure interaction term of the variables to its left and right in a model.	$Y \sim X_1 : X_2$ will define the model $Y = \beta_0 + \beta_1 X_1 X_2$.
**	Use this operator to define a model that includes the nth order of its interaction terms. n here is an integer. See the example for details.	$Y \sim (X_1 + X_2 + X_3)^{**2}$ will define the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_1 X_2 + \beta_5 X_1 X_3 + \beta_6 X_2 X_3$.
f(expr)	This denotes the possibility to include an arbitrary Python function f(expr) in a Patsy formula string.	$Y \sim X_1 + np.log(X_2)$ will define the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 np.log(X_2)$. Note that f(expr) is equal to np.log(X_2).
I(expr)	Use this to escape operators in an expression f(expr) so that they have arithmetic meaning rather than the set operator meaning used by the Patsy formula parser.	$Y \sim X_1 + np.log(I(X_2 + X_3))$ will define a model where the plus operator in $(X_2 + X_3)$ is treated as an arithmetic plus sign by the Patsy formula parser because of the identity operator (denoted as I()), and NOT as a set union of X_2 and X_3 .
C(var)	Use this to define a variable (var) in a Patsy formula string as a categorical variable type.	Defining C(X) in a formula will tell the formula parser to treat the values of the term as categorical,

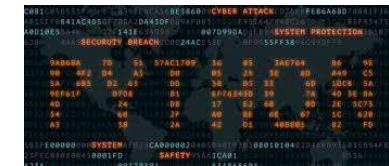
Intercept Handling

- In patsy formula strings, string 1 is used to define the intercept of a model.
- Because the intercept is needed most of the time, string 1 is automatically included in every formula string definition.
- You don't have to include it in your string to specify the intercept. It is there invisibly.

Complete Activity 2.01: Fitting a Log-Linear Model Using the Statsmodels Formula API

Multiple Regression Analysis

- In the exercises and activity so far, we have used only one independent variable in our regression analysis.
 - In practice, as we have seen with the Boston Housing dataset, processes and phenomena of analytic interest are rarely influenced by only one feature.



Complete Exercise 2.05: Fitting a Multiple Linear Regression Model Using the Statsmodels Formula API

Assumptions of Regression Analysis

- Due to the parametric nature of linear regression analysis, the method makes certain assumptions about the data it analyzes.
- When these assumptions are not met, the results of the regression analysis may be misleading to say the least.
- It is, therefore, necessary to check any analysis work to ensure the regression assumptions are not violated.

Complete Activity 2.02: Fitting a Multiple Log-Linear Regression Model

Explaining the Results of Regression Analysis

- A primary objective of regression analysis is to find a model that explains the variability observed in a dependent variable of interest.
- It is, therefore, very important to have a quantity that measures how well a regression model explains this variability.



Regression Analysis Checks and Balances

- In the preceding discussions, we used the R-squared and the Adjusted R-squared statistics to assess the goodness of fit of our models.
- While the R-squared statistic provides an estimate of the strength of the relationship between a model and the dependent variable(s), it does not provide a formal statistical hypothesis test for this relationship.

The F-test

- The F-test is what validates the overall statistical significance of the strength of the relationship between a model and its dependent variables.
- If the p-value for the F-test is less than the chosen α -level (0.05, in our case), we reject the null hypothesis and conclude that the model is statistically significant overall.

The t-test

- Once a model has been determined to be statistically significant globally, we can proceed to examine the significance of individual independent variables in the model.
- In Figure, the p-values (denoted $p>|t|$ in Section 2) for the independent variables are provided.
- The p-values were calculated using the t-values also given on the summary results.

Summary

- This lesson introduced the topic of linear regression analysis using Python.
- We learned that regression analysis, in general, is a supervised machine learning or data science problem.
- We learned about the fundamentals of linear regression analysis, including the ideas behind the method of least squares.
- We also learned about how to use the pandas Python module to load and prepare data for exploration and analysis.



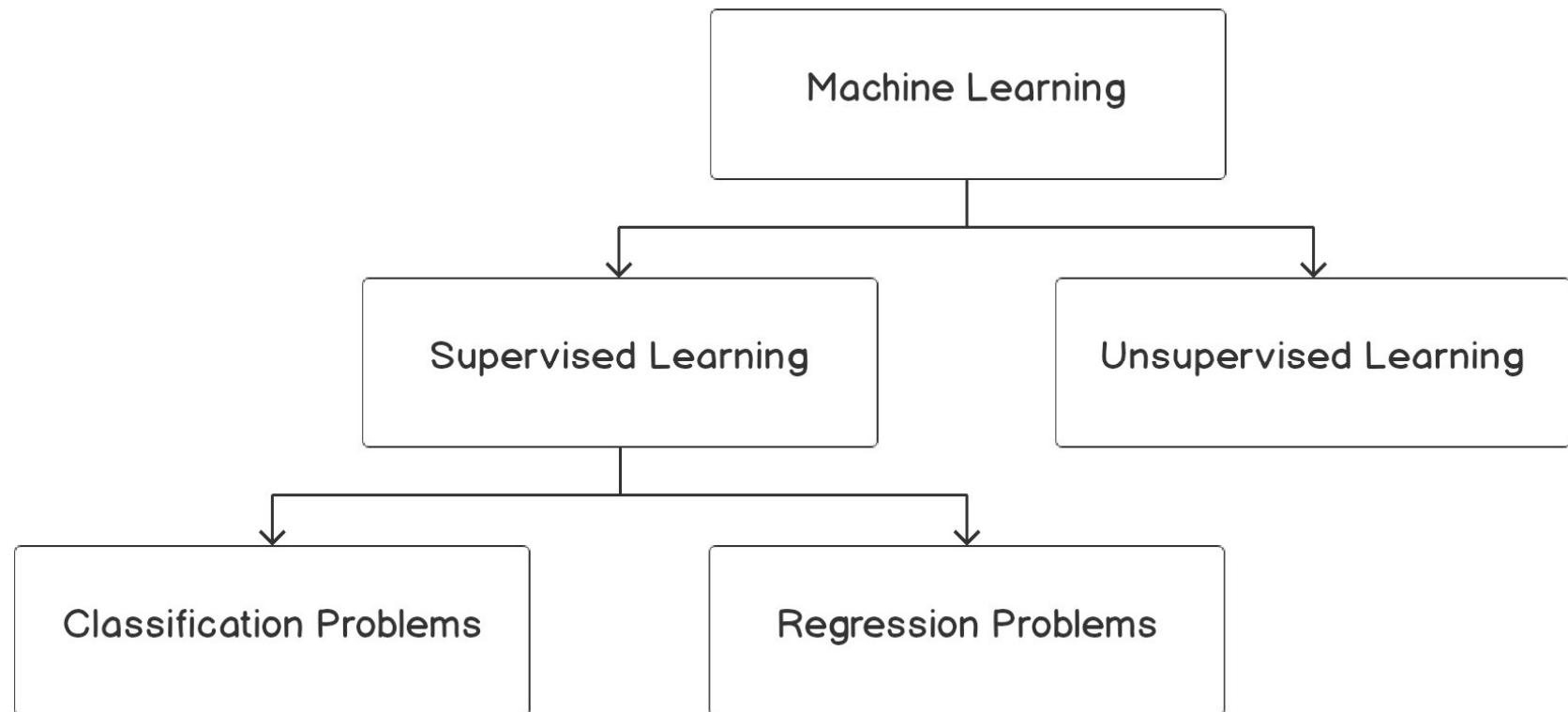
3. Binary Classification



Overview

- In this lesson, we will be using a real-world dataset and a supervised learning technique called classification to generate business outcomes.
- By the end of this lesson, you will be able to formulate a data science problem statement from a business perspective

Introduction



Introduction

- Predicting whether a customer will buy the recommended product
- Identifying whether a credit transaction is fraudulent
- Determining whether a patient has a disease
- Analyzing images of animals and predicting whether the image is of a dog, cat, or panda
- Analyzing text reviews and capturing the underlying emotion such as happiness, anger, sorrow, or sarcasm

Introduction

If you observe the preceding examples, there is a subtle difference between the first three and the last two. The first three revolve around binary decisions:

- Customers can either buy the product or not.
- Credit card transactions can be fraudulent or legitimate.
- Patients can be diagnosed as positive or negative for a disease.

Understanding the Business Context

- The best way to work using a concept is with an example you can relate to.
- To understand the business context, let's, for instance, consider the following example.
- The marketing head of the bank where you are a data scientist approaches you with a problem they would like to be addressed.

Business Discovery

- The first process when embarking on a data science problem like the preceding is the business discovery process.
- This entails understanding various drivers influencing the business problem.
- Getting to know the business drivers is important as it will help in formulating hypotheses about the business problem, which can be verified during the exploratory data analysis (EDA).

Business Discovery

- Would age be a factor, with more propensity shown by the elderly?
- Is there any relationship between employment status and the propensity to buy term deposits?
- Would the asset portfolio of a customer—that is, house, loan, or higher bank balance— influence the propensity to buy?
- Will demographics such as marital status and education influence the propensity to buy term deposits? If so, how are demographics correlated to a propensity to buy?

Complete Exercise 3.01: Loading and Exploring the Data from the Dataset

Testing Business Hypotheses Using Exploratory Data Analysis

- In the previous section, you approached the problem statement from a domain perspective, thereby identifying some of the business drivers.
- Once business drivers are identified, the next step is to evolve some hypotheses about the relationship of these business drivers and the business outcome you have set out to achieve.



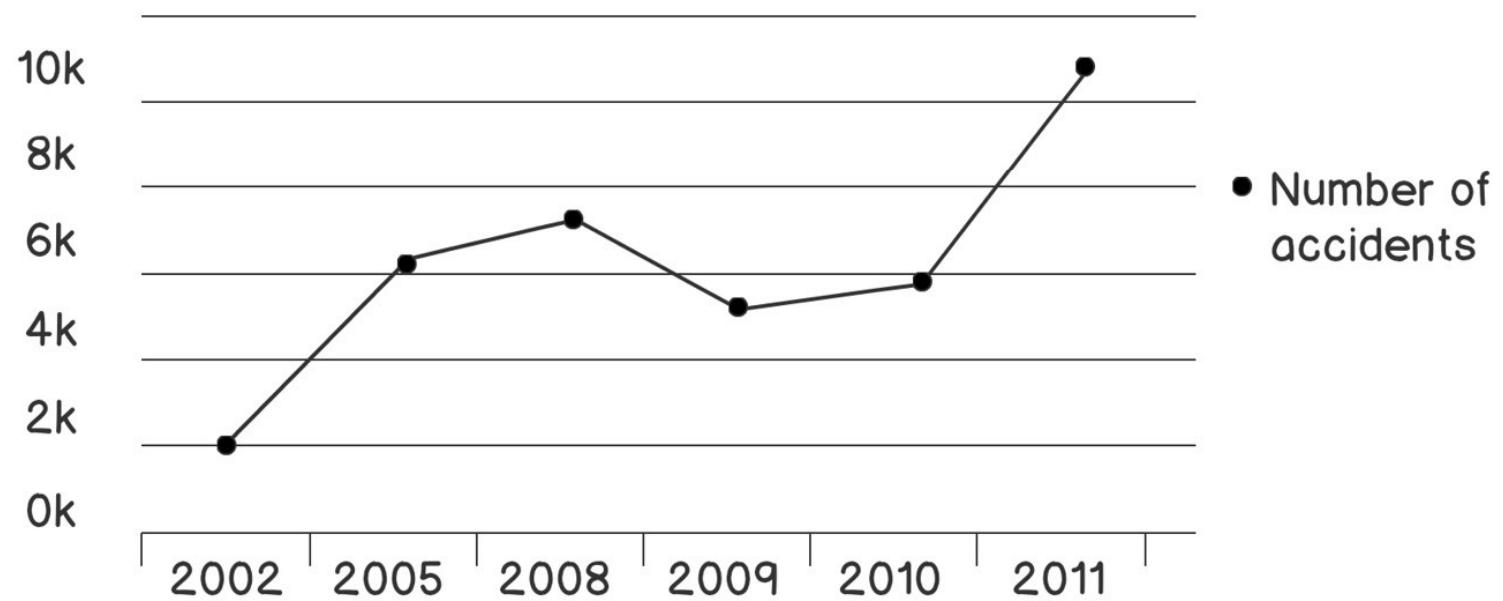
Visualization for Exploratory Data Analysis

- Visualization is imperative for EDA.
- Effective visualization helps in deriving business intuitions from the data.

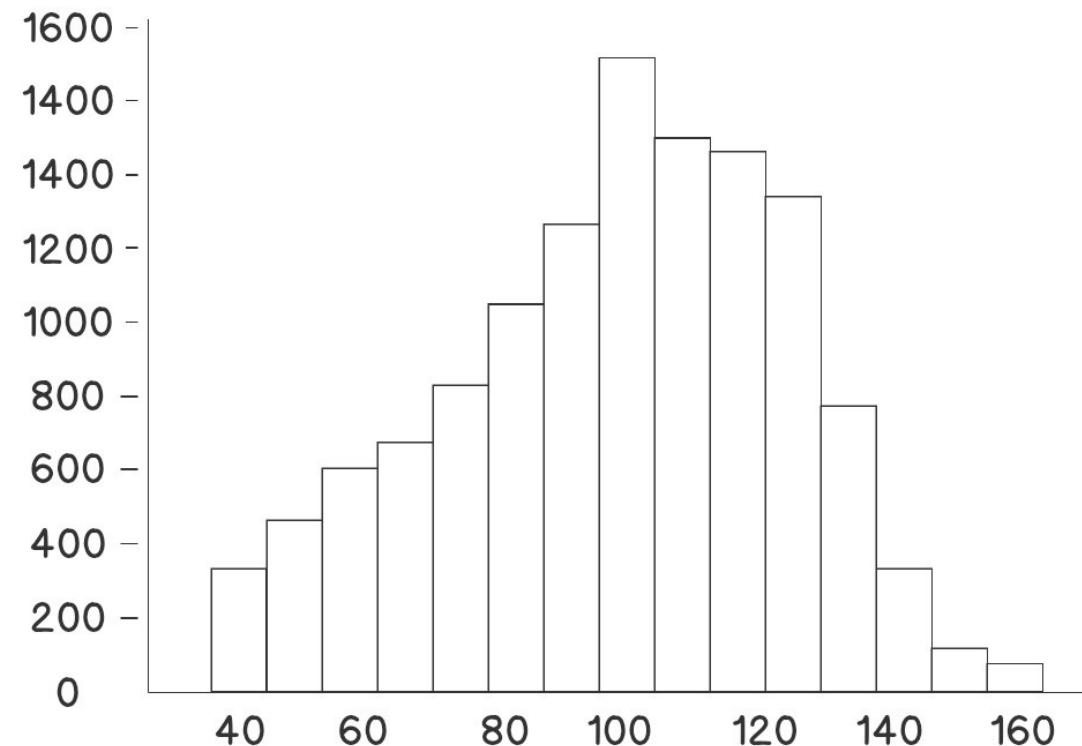


```
33     self.debug = True
34     self.logpath = None
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(path, 'w')
39         self.file.write('')
40         self.fingerprints = {}
41     else:
42         self.fingerprints = {}
43     classmethod
44     def from_settings(cls, settings):
45         debug = settings.getboolean('DEBUG')
46         return cls(settings)
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints[fp] = request
52         if self.file:
```

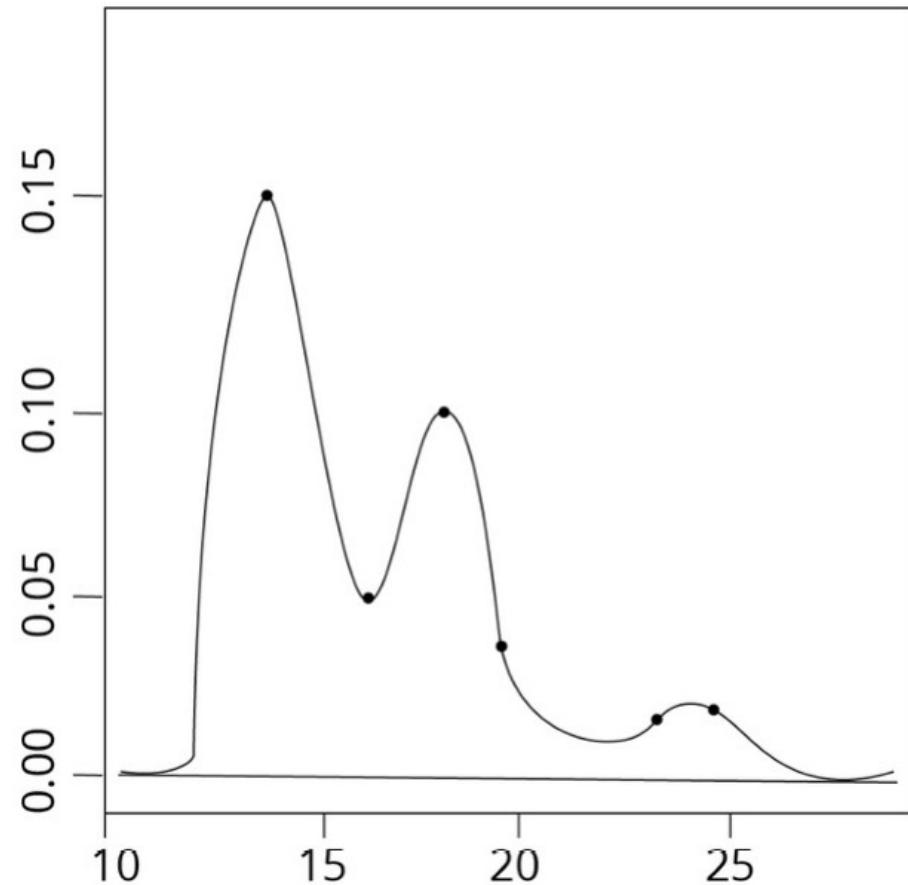
Line graphs



Histograms



Density plots



Stacked bar charts

- A stacked bar chart helps you to visualize the various categories of data, one on top of the other, in order to give you a sense of proportion of the categories; for instance, if you want to plot a bar chart showing the values, Yes and No, on a single bar.
- This can be done using the stacked bar chart, which cannot be done on the other charts.

Stacked bar charts

- Import the library files required for the task:

```
# Importing library files
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

- Next, create some sample data detailing a list of jobs:

```
# Create a simple list of categories
```

```
jobList = ['admin','scientist','doctor','management']
```

Stacked bar charts

- Each job will have two categories to be plotted, yes and No, with some proportion between yes and No.
- These are detailed as follows:

```
# Getting two categories ( 'yes','No') for each of jobs  
jobYes = [20,60,70,40]  
jobNo = [80,40,30,60]
```

Stacked bar charts

- Next, let's define the width of each bar and do the plotting.
- In the plot, p2, we define that when stacking, Yes will be at the bottom and No at top:

```
# Get width of each bar
```

```
width = 0.35
```

```
# Getting the plots
```

```
p1 = plt.bar(ind, jobYes, width)
```

```
p2 = plt.bar(ind, jobNo, width, bottom=jobYes)
```

Stacked bar charts

- Define the labels for the Y axis and the title of the plot:

```
# Getting the labels for the plots  
plt.ylabel('Proportion of Jobs')  
plt.title('Job')
```



```
33     self.logger = None  
34     self.logpath = None  
35     self.debug = False  
36     self.logger = logging.getLogger('Job')  
37     if path:  
38         self.path = os.path.dirname(path)  
39         self.file = os.path.basename(path)  
40         self.fingerprints = set()  
41  
42     *classmethod  
43     def from_settings(cls, settings):  
44         debug = settings.getboolean('logger.debug')  
45         return cls(debug=debug, settings=settings)  
46  
47     def request_seemself(self, request):  
48         fp = self.request_fingerprint(request)  
49         if fp in self.fingerprints:  
50             return True  
51         self.fingerprints.add(fp)  
52         self.logger.info('seen %s', fp)
```

Stacked bar charts

- The indexes for the X and Y axes are defined next.
- For the X axis, the list of jobs are given, and, for the Y axis, the indices are in proportion from 0 to 100 with an increment of 10 (0, 10, 20, 30, and so on):

```
# Defining the x label indexes and y label indexes  
plt.xticks(ind, jobList)  
plt.yticks(np.arange(0, 100, 10))
```

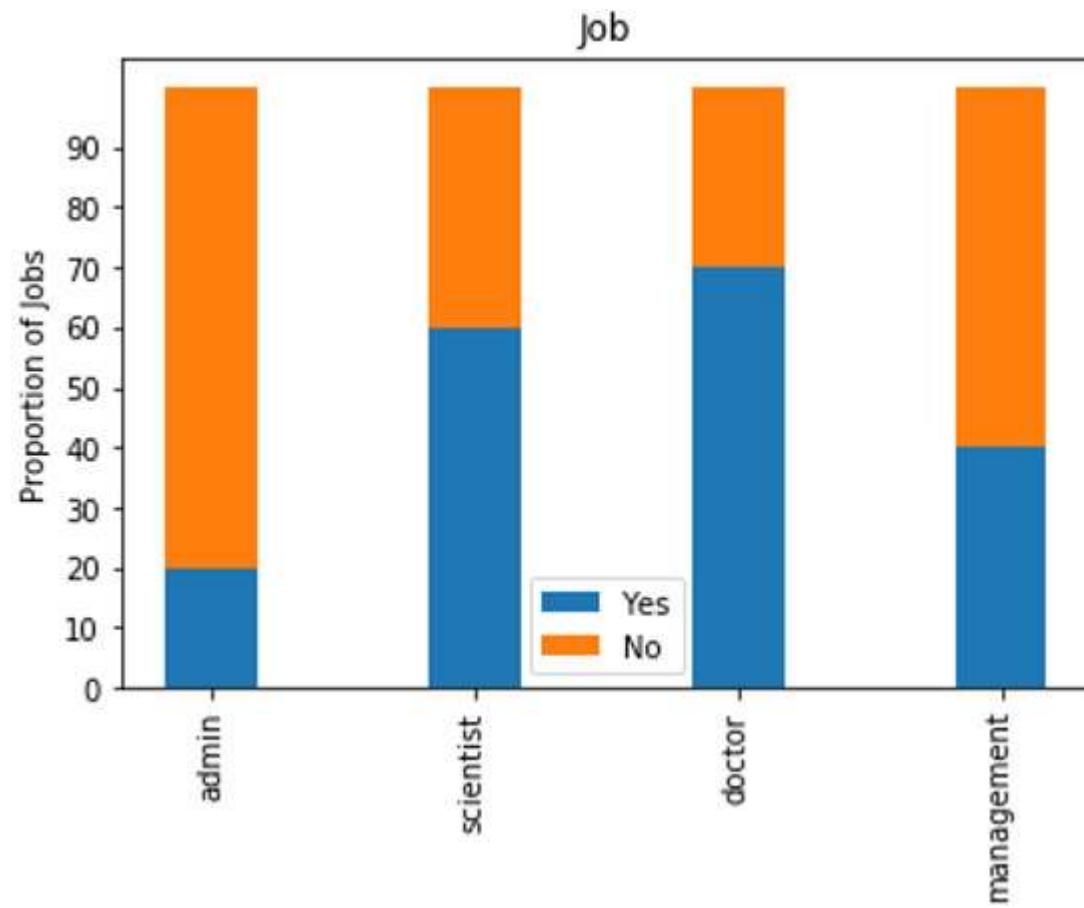
Stacked bar charts

- The last step is to define the legends and to rotate the axis labels to 90 degrees.
- The plot is finally displayed:

```
# Defining the legends
plt.legend((p1[0], p2[0]), ('Yes', 'No'))
# To rotate the axis labels
plt.xticks(rotation=90)
plt.show()
```



```
31:         self.debug = True
32:         self.logups = True
33:         self.debug = debug
34:         self.logger = logging.getLogger('Client')
35:         if path:
36:             self.file = os.path.join(path, 'client.log')
37:             self.file.level = logging.INFO
38:             self.fingerprints = {}
39:         else:
40:             self.fingerprints = {}
41:         self.settings = None
42:         self.settings = settings
43:         self.debug = settings.debug
44:         self.logups = settings.logups
45:         self.file = settings.logfile
46:         self.fingerprints = settings.fingerprints
47:     def from_settings(cls, settings):
48:         debug = settings.debug
49:         logups = settings.logups
50:         path = settings.logpath
51:         return cls(path=path, debug=debug, logups=logups)
```



Complete Exercise 3.02: Business Hypothesis Testing for Age versus Propensity for a Term Loan

Intuitions from the Exploratory Analysis

- What are the intuitions we can take out of the exercise that we have done so far? We have seen two contrasting plots by taking the proportion of users and without taking the proportions.
- As you can see, taking the proportion of users is the right approach to get the right perspective in which we must view data.
- This is more in line with the hypothesis that we have evolved.

Complete Activity 3.01: Business Hypothesis Testing to Find Employment Status versus Propensity for Term Deposits

Feature Engineering

There are two broad types of feature engineering:

1. Here, we transform raw variables based on intuitions from a business perspective. These intuitions are what we build during the exploratory analysis.
2. The transformation of raw variables is done from a statistical and data normalization perspective.

Business-Driven Feature Engineering

- Business-driven feature engineering is the process of transforming raw variables based on business intuitions that were derived during the exploratory analysis.
- It entails transforming data and creating new variables based on business factors or drivers that influence a business problem.

Complete Exercise 3.03: Feature Engineering – Exploration of Individual Features

Complete Exercise 3.04: Feature Engineering – Creating New Features from Existing Ones

Data-Driven Feature Engineering

A Quick Peek at Data Types and a Descriptive Summary

- Looking at the data types such as categorical or numeric and then deriving summary statistics is a good way to take a quick peek into data before you do some of the downstream feature engineering steps.
- Let's take a look at an example from our dataset:

```
# Looking at Data types  
print(bankData.dtypes)  
# Looking at descriptive statistics  
print(bankData.describe())
```

age	int64
job	object
marital	object
education	object
default	object
balance	int64
housing	object
loan	object
contact	object
day	int64
month	object
duration	int64
campaign	int64
pdays	int64
previous	int64
poutcome	object
y	object
balanceClass	object
balanceTran	float64
loanTran	int64
houseTran	int64
assetIndex	float64
assetClass	object

	age	balance	day	duration	campaign	\
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	
std	10.618762	3044.765829	8.322476	257.527812	3.098021	
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	
25%	33.000000	72.000000	8.000000	103.000000	1.000000	
50%	39.000000	448.000000	16.000000	180.000000	2.000000	
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	

	pdays	previous	balanceTran	loanTran	houseTran	\
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	
mean	40.197828	0.580323	0.085171	4.359094	3.223353	
std	100.128746	2.303441	0.027643	1.467280	1.987511	
min	-1.000000	0.000000	0.000000	1.000000	1.000000	
25%	-1.000000	0.000000	0.073457	5.000000	1.000000	
50%	-1.000000	0.000000	0.076871	5.000000	5.000000	
75%	-1.000000	0.000000	0.085768	5.000000	5.000000	
max	871.000000	275.000000	1.000000	5.000000	5.000000	

	assetIndex					
count	45211.000000					
mean	1.179050					
std	0.951987					
min	0.000000					
25%	0.376636					
50%	0.569154					
75%	1.902475					
max	15.107902					

Correlation Matrix and Visualization

- Correlation, as you know, is a measure that indicates how two variables fluctuate together.
- Any correlation value of 1, or near 1, indicates that those variables are highly correlated.
- Highly correlated variables can sometimes be damaging for the veracity of models and, in many circumstances, we make the decision to eliminate such variables or to combine them to form composite or interactive variables.

Complete Exercise 3.05: Finding the Correlation in Data to Generate a Correlation Plot Using Bank Data

Skewness of Data

- Another area for feature engineering is skewness.
- Skewed data means data that is shifted in one direction or the other.
- Skewness can cause machine learning models to underperform.
- Many machine learning models assume normally distributed data or data structures to follow the Gaussian structure.

Skewness of Data

- Let's take a look at the following example.
- Here, we use the `.skew()` function to find the skewness in data.
- For instance, to find the skewness of data in our `bank-full.csv` dataset, we perform the following:

```
# Skewness of numeric attributes  
bankNumeric.skew()
```

Skewness of Data

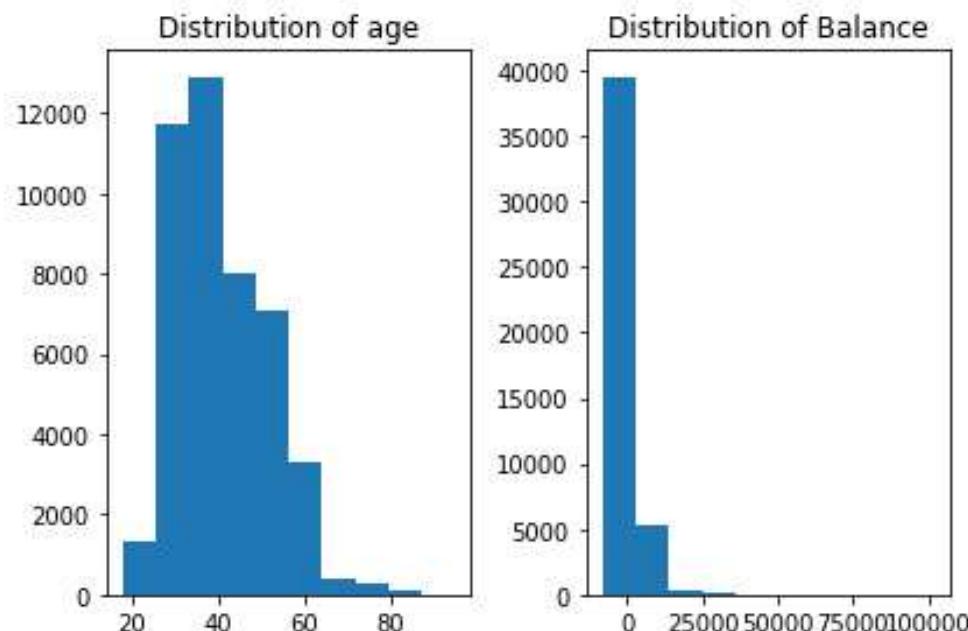
```
age           0.685
balance       8.360
day            0.093
duration      3.144
campaign      4.899
pdays          2.616
previous      41.846
dtype: float64
```

Histograms

```
# Histograms
from matplotlib import pyplot as plt
fig, axs = plt.subplots(1,2)
axs[0].hist(bankNumeric['age'])
axs[0].set_title('Distribution of age')
axs[1].hist(bankNumeric['balance'])
axs[1].set_title('Distribution of Balance')
# Ensure plots do not overlap
plt.tight_layout()
```

Histograms

- You should get this output:

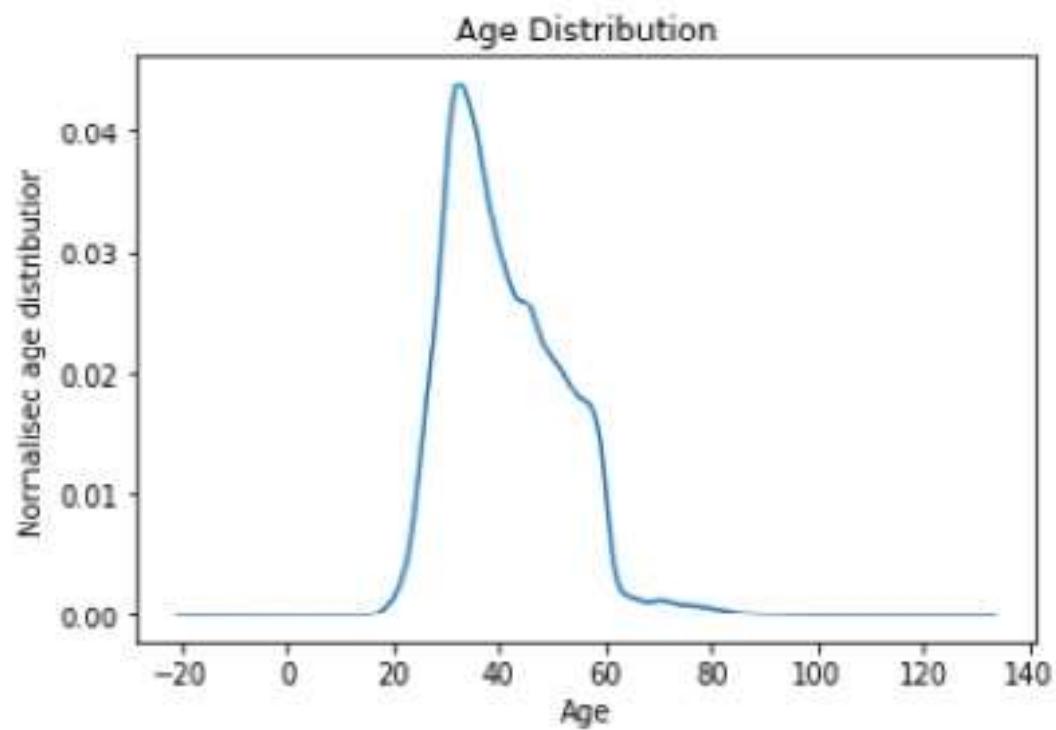


Density Plots

- Density plots help in visualizing the distribution of data.
- A density plot can be created using the kind = 'density' parameter:

```
from matplotlib import pyplot as plt
# Density plots
bankNumeric['age'].plot(kind = 'density', subplots = False, \
                        layout = (1,1))
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Normalised age distribution')
pyplot.show()
```

Density Plots



Other Feature Engineering Methods

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from numpy import set_printoptions
scaling = StandardScaler().fit(bankNumeric)
rescaledNum = scaling.transform(bankNumeric)
set_printoptions(precision = 3)
print(rescaledNum)
```

Other Feature Engineering Methods

- You should get this output:

```
[[ 1.607  0.256 -1.298 ...  0.437  0.894  1.184]
 [ 0.289 -0.438 -1.298 ...  0.437  0.894  0.68 ]
 [-0.747 -0.447 -1.298 ... -2.289  0.894 -0.856]
 ...
 [ 2.925  1.43   0.143 ...  0.437 -1.119 -0.584]
 [ 1.513 -0.228  0.143 ...  0.437 -1.119 -0.824]
 [-0.371  0.528  0.143 ...  0.437 -1.119 -0.714]]
```

Other Feature Engineering Methods

- The following code uses the normalizer data transmission techniques:

```
# Normalizing Data (Length of 1)
from sklearn.preprocessing import Normalizer
normaliser = Normalizer().fit(bankNumeric)
normalisedNum = normaliser.transform(bankNumeric)
set_printoptions(precision = 3)
print(normalisedNum)
```

Other Feature Engineering Methods

- You should get this output:

```
[[2.686e-02 9.923e-01 2.315e-03 ... 2.315e-03 2.315e-03 1.068e-03]
 [2.747e-01 1.810e-01 3.121e-02 ... 3.121e-02 3.121e-02 1.140e-02]
 [3.966e-01 2.404e-02 6.010e-02 ... 1.202e-02 6.010e-02 4.376e-03]
 ...
 [1.235e-02 9.805e-01 2.917e-03 ... 8.579e-04 1.716e-04 1.070e-04]
 [6.775e-02 7.940e-01 2.021e-02 ... 5.943e-03 1.189e-03 4.687e-04]
 [1.234e-02 9.906e-01 5.668e-03 ... 1.667e-03 3.334e-04 1.663e-04]]
```

Summarizing Feature Engineering

- We obtain intuitions from a business perspective through EDA
- Based on the business intuitions, we devised a new feature that is a combination of three other variables.
- We verified the influence of constituent variables of the new feature and devised an approach for weights to be applied.
- Converted ordinal data into corresponding weights.

Building a Binary Classification Model Using the Logistic Regression Function

- By now, you may have realized that the goal of machine learning is to estimate a mapping function (f) between an output variable and input variables.
- In mathematical form, this can be written as follows:

$$Y = f(X)$$

Building a Binary Classification Model Using the Logistic Regression Function

- For simplicity, let's assume that we have only two attributes, age and bank balance.
- Using these, we have to predict whether a customer is likely to buy a term deposit or not.
- Let the age be 40 years and the balance \$1,000. With all of these attribute values, let's assume that the mapping equation is as follows:

$$Y = M_0 + M_1 \text{age} * \text{Age} + M_2 \text{balance} * \text{balance}$$

Building a Binary Classification Model Using the Logistic Regression Function

- To understand the concept better, let's take the example of a linear model.
- For a linear model, the mapping function takes the following form:

$$Y = C_0 + M_1 * X_1 + M_2 * X_2$$

Logistic Regression Demystified

- Logistic regression is a linear model similar to the linear regression that was covered in the previous lesson.
- The mathematical equation for a logistic regression function can be written as follows:

$$Y = \frac{e^{(C_0 + M_1 * X_1 + M_2 * X_2)}}{(1 + e^{(C_0 + M_1 * X_1 + M_2 * X_2)})}$$

Logistic Regression Demystified

- To transform the real-valued output into a probability, we use the logistic function, which has the following form:

$$\frac{e^x}{1 + e^x}$$

Logistic Regression Demystified

- In the business problem, we are trying to predict the probability of whether a customer would buy a term deposit or not.
- To do that, let's return to the example we derived from the problem statement:

$$Y = M_0 + M_{1\text{age}} * \text{Age} + M_{2\text{balance}} * \text{balance}$$

Logistic Regression Demystified

- Adding the following values, we get $Y = 0.1 + 0.4 * 40 + 0.002 * 100$.
- To get the probability, we must transform this problem statement using the logistic function, as follows:

$$Y = \frac{e^{(0.1 + 0.4 \times 40 + 0.002 \times 1000)}}{(1 + e^{(0.1 + 0.4 \times 40 + 0.002 \times 1000)})}$$

Confusion Matrix

Test Examples	Ground Truth	Predictions	Evaluation
Example 1	Yes	Yes	Correct
Example 2	Yes	No	Misclassified
Example 3	Yes	Yes	Correct
Example 4	No	No	Correct
Example 5	Yes	Yes	Correct
Example 6	No	Yes	Misclassified
Example 7	Yes	No	Misclassified

Confusion Matrix

- A confusion matrix generates the resultant comparison between prediction and ground truth, as represented in the following table:

Ground truth	Predicted	
	Yes	No
Yes	True positive (TP) = 3	False negative (FN) = 2
No	False positive (FP) = 1	True negative (TN) = 1

Accuracy

- Accuracy is the first level of evaluation, which we will resort to in order to have a quick check on model performance.
- Referring to the preceding table, accuracy can be represented as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Classification Report

- A classification report outputs three key metrics: precision, recall, and the F1 score.
- Precision is the ratio of true positives to the sum of true positives and false positives:

$$\frac{tp}{(tp + fp)}$$

Classification Report

- Precision is the indicator that tells you, out of all of the positives that were predicted, how many were true positives.
- Recall is the ratio of true positives to the sum of true positives and false negatives:

$$\frac{tp}{(tp + fn)}$$

Data Preprocessing

- Data preprocessing has an important role to play in the life cycle of data science projects.
- These processes are often the most time-consuming part of the data science life cycle.
- Careful implementation of the preprocessing steps is critical and will have a strong bearing on the results of the data science project.

Complete Exercise 3.06: A Logistic Regression Model for Predicting the Propensity of Term Deposit Purchases in a Bank

Complete Activity 3.02: Model Iteration 2 – Logistic Regression Model with Feature Engineered Variables

Summary

- In this lesson, we learned about binary classification using logistic regression from the perspective of solving a use case.
- Let's summarize our learnings in this lesson.
- We were introduced to classification problems and specifically binary classification problems.



4. Multiclass Classification with RandomForest



Overview

- This lesson will show you how to train a multiclass classifier using the Random Forest algorithm.
- You will also see how to evaluate the performance of multiclass models.

Introduction

- In the data science industry, quite often you will face multiclass classification problems.
- For example, if you were working for Netflix or any other streaming platform, you would have to build a model that could predict the user rating for a movie based on key attributes such as genre, duration, or cast.
- A potential list of rating values may be: Hate it, Dislike it, Neutral, Like it, Love it.

Training a Random Forest Classifier

- The Random Forest methodology was first proposed in 1995 by Tin Kam Ho but it was first developed by Leo Breiman in 2001.
- So Random Forest is not really a recent algorithm per se and It has been in use for almost two decades already.
- But its popularity hasn't faded, thanks to its performance and simplicity.

Training a Random Forest Classifier

- First, we need to load the data from the GitHub repository using pandas and then we will print its first five rows using the head() method.

```
import pandas as pd  
file_url = 'https://raw.githubusercontent.com/fenago'\  
    '/data-science/master/Lab04/'\  
    'Dataset/activity.csv'  
df = pd.read_csv(file_url)  
df.head()
```

Training a Random Forest Classifier

	avg_rss12	var_rss12	avg_rss13	var_rss13	avg_rss23	var_rss23	Activity
0	42.00	0.00	18.50	0.50	12.00	0.00	bending1
1	42.00	0.00	18.00	0.00	11.33	0.94	bending1
2	42.75	0.43	16.75	1.79	18.25	0.43	bending1
3	42.50	0.50	16.75	0.83	19.00	1.22	bending1
4	43.00	0.82	16.25	0.83	18.00	0.00	bending1

Training a Random Forest Classifier

- In this example, you will accurately predict the target variable ('Activity') from the features (the six other columns) using Random Forest.
- For example, for the first row of the preceding example, the model will receive the following features as input and will predict the 'bending1' class:

	avg_rss12	var_rss12	avg_rss13	var_rss13	avg_rss23	var_rss23	Activity
0	42.00	0.00	18.50	0.50	12.00	0.00	bending1

Training a Random Forest Classifier

- The `.pop()` method extracts the specified column and removes it from the DataFrame:

```
target = df.pop('Activity')
```

Training a Random Forest Classifier

- We need to specify the following parameters for this function: the feature and target variables, the ratio of the testing set (`test_size`), and `random_state` in order to get reproducible results if we have to run the code again:

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split\  
    (df, target, test_size=0.33, \  
     random_state=42)
```

Training a Random Forest Classifier

- Now that we have got our training and testing sets, we are ready for modeling.
- Let's first import the RandomForestClassifier class from sklearn.ensemble:

```
from sklearn.ensemble import RandomForestClassifier
```

Training a Random Forest Classifier

- We will walk you through some of the key hyperparameters in the following sections:

```
rf_model = RandomForestClassifier(random_state=1,  
                                  n_estimators=10)
```

Training a Random Forest Classifier

- We need to specify the features and target variables as parameters:

```
rf_model.fit(X_train, y_train)
```

- The output will be as follows:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=1, verbose=0,
                      warm_start=False)
```

Training a Random Forest Classifier

- Now that the model has completed its training, we can use the parameters it learned to make predictions on the input data we will provide.
- In the following example, we are using the features from the training set:

```
preds = rf_model.predict(X_train)
```

Training a Random Forest Classifier

- Now we can print these predictions:
`preds`
- The output will be as follows:

```
array(['lying', 'bending1', 'cycling', ..., 'cycling', 'bending1',
       'standing'], dtype=object)
```

Evaluating the Model's Performance

- In this lesson, we will use a metric called accuracy score.
- It calculates the ratio between the number of correct predictions and the total number of predictions made by the model:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

Evaluating the Model's Performance

- For instance, if your model made 950 correct predictions out of 1,000 cases, then the accuracy score would be $950/1000 = 0.95$.
- This would mean that your model was 95% accurate on that dataset.
- The sklearn package provides a function to calculate this score automatically and it is called `accuracy_score()`, We need to import it first:
`from sklearn.metrics import accuracy_score`

Evaluating the Model's Performance

- We will reuse the predictions from the previous section – preds:

`accuracy_score(y_train, preds)`

- The output will be as follows:

`0.9876688826935449`

Evaluating the Model's Performance

- Let's calculate the accuracy score for the testing set:

```
test_preds = rf_model.predict(X_test)  
accuracy_score(y_test, test_preds)
```

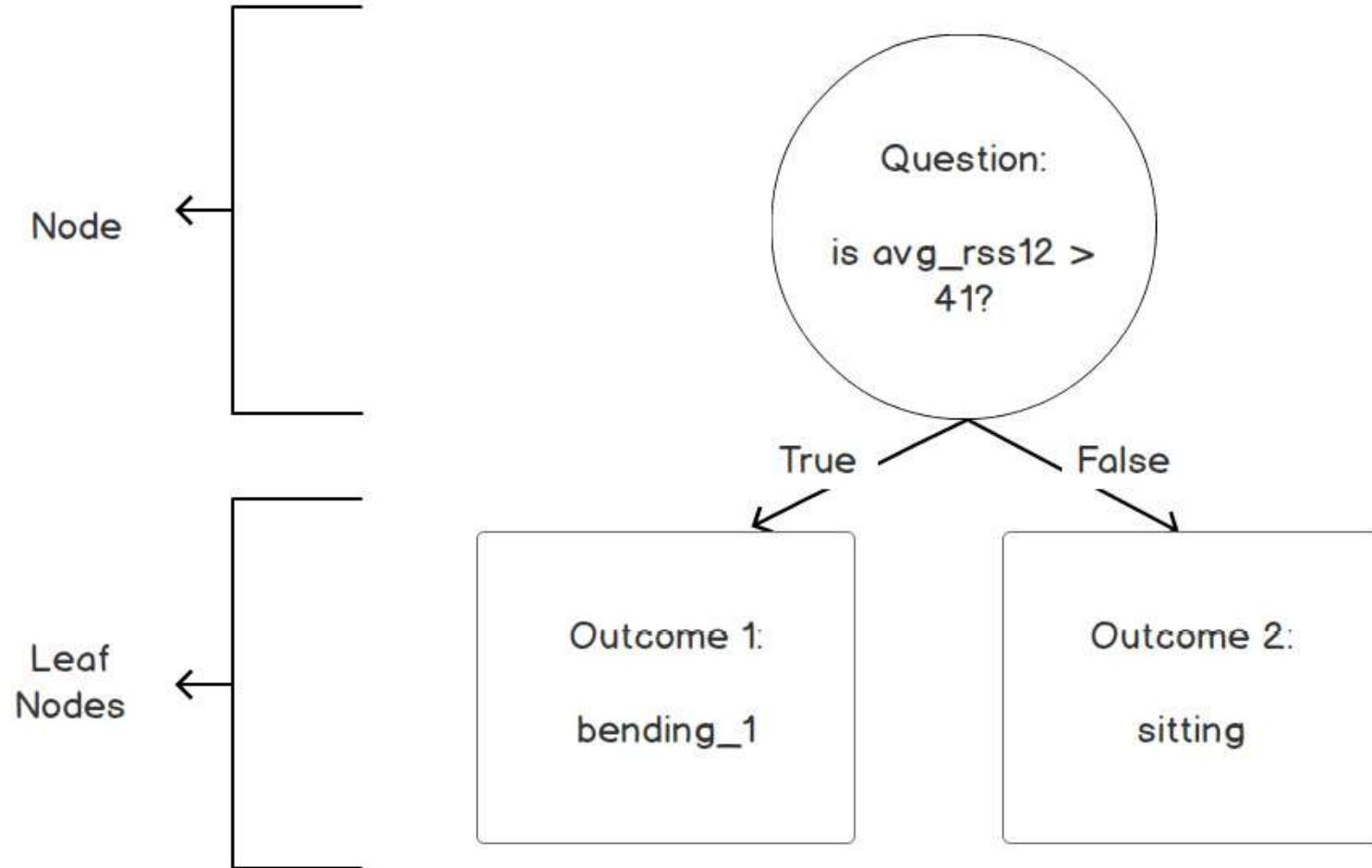
- The output will be as follows:

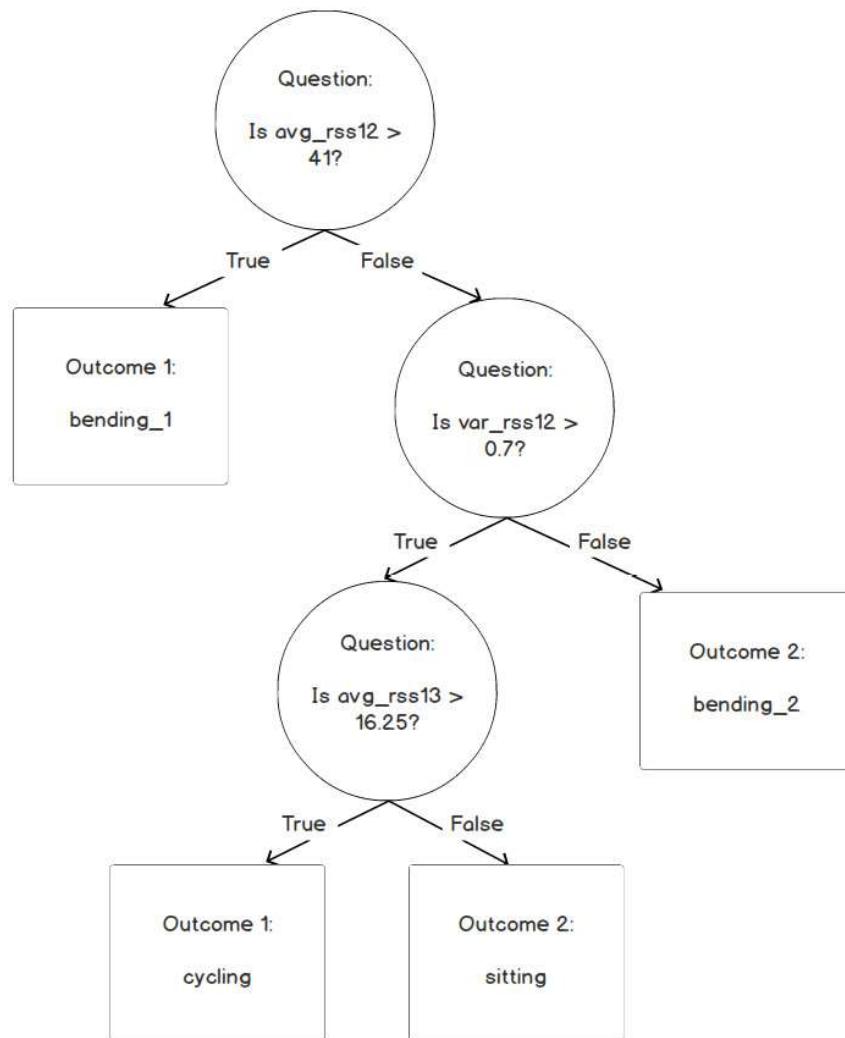
0.767868804876279

Complete Exercise 4.01: Building a Model for Classifying Animal Type and Assessing Its Performance

Number of Trees Estimator

- A tree is a logical graph that maps a decision and its outcomes at each of its nodes.
- Simply speaking, it is a series of yes/no (or true/false) questions that lead to different outcomes.
- A leaf is a special type of node where the model will make a prediction.
- There will be no split after a leaf & A single node split of a tree may look like(next slide).





Number of Trees Estimator

```
rf_model2 = RandomForestClassifier(random_state=1, \
                                    n_estimators=2)
rf_model2.fit(X_train, y_train)
preds2 = rf_model2.predict(X_train)
test_preds2 = rf_model2.predict(X_test)
print(accuracy_score(y_train, preds2))
print(accuracy_score(y_test, test_preds2))
```

- The output will be as follows:

```
0.8881978697548073
0.6971917857920326
```

Number of Trees Estimator

```
rf_model3 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50)
rf_model3.fit(X_train, y_train)
preds3 = rf_model3.predict(X_train)
test_preds3 = rf_model3.predict(X_test)
print(accuracy_score(y_train, preds3))
print(accuracy_score(y_test, test_preds3))
```

- The output will be as follows:

```
0.9969618986346415
0.7897830346128728
```

Complete Exercise 4.02: Tuning n_estimators to Reduce Overfitting

Maximum Depth

- Let's try several values for this hyperparameter on the Activity Recognition dataset: 3, 10, and 50:

```
rf_model4 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, max_depth=3)
rf_model4.fit(X_train, y_train)
preds4 = rf_model4.predict(X_train)
test_preds4 = rf_model4.predict(X_test)
print(accuracy_score(y_train, preds4))
print(accuracy_score(y_test, test_preds4))
```

- You should get the following output:

```
0.6133747944813782
0.6122922864813874
```

Maximum Depth

- Let's increase max_depth to 10:

```
rf_model5 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, \
                                    max_depth=10)

rf_model5.fit(X_train, y_train)
preds5 = rf_model5.predict(X_train)
test_preds5 = rf_model5.predict(X_test)
print(accuracy_score(y_train, preds5))
print(accuracy_score(y_test, test_preds5))

0.811423261133748
0.7637326754226834
```

Maximum Depth

```
rf_model6 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, \
                                    max_depth=50)
rf_model6.fit(X_train, y_train)
preds6 = rf_model6.predict(X_train)
test_preds6 = rf_model6.predict(X_test)
print(accuracy_score(y_train, preds6))
print(accuracy_score(y_test, test_preds6))
```

- The output will be as follows:

```
0.9969618986346415
0.7897830346128728
```

Complete Exercise 4.03: Tuning `max_depth` to Reduce Overfitting

Minimum Sample in Leaf

- Let's try to find the optimal value for min_samples_leaf for the Activity Recognition dataset:

```
rf_model7 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, \
                                    max_depth=10, \
                                    min_samples_leaf=3)

rf_model7.fit(X_train, y_train)
preds7 = rf_model7.predict(X_train)
test_preds7 = rf_model7.predict(X_test)
print(accuracy_score(y_train, preds7))
print(accuracy_score(y_test, test_preds7))
```

- The output will be as follows:

```
0.8037029094288369
0.7611929468108265
```

Minimum Sample in Leaf

```
rf_model8 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, \
                                    max_depth=10, \
                                    min_samples_leaf=10)

rf_model8.fit(X_train, y_train)
preds8 = rf_model8.predict(X_train)
test_preds8 = rf_model8.predict(X_test)
print(accuracy_score(y_train, preds8))
print(accuracy_score(y_test, test_preds8))
```

- The output will be as follows:

```
0.7930159410965759
0.7623539656048183
```

Minimum Sample in Leaf

```
rf_model9 = RandomForestClassifier(random_state=1, \
                                    n_estimators=50, \
                                    max_depth=10, \
                                    min_samples_leaf=25)

rf_model9.fit(X_train, y_train)
preds9 = rf_model9.predict(X_train)
test_preds9 = rf_model9.predict(X_test)
print(accuracy_score(y_train, preds9))
print(accuracy_score(y_test, test_preds9))
```

- The output will be as follows:

```
0.7810422474801629
0.7593062912705899
```

Complete Exercise 4.04: Tuning min_samples_leaf

Maximum Features

- We are getting close to the end of this lesson, You have already learned how to tune several of the most important hyperparameters for RandomForest.
- In this section, we will present you with another extremely important one: `max_features`.
- Earlier, we learned that RandomForest builds multiple trees and takes the average to make predictions.
- This is why it is called a forest, but we haven't really discussed the "random" part yet

Maximum Features

```
rf_model10 = RandomForestClassifier(random_state=1, \
                                     n_estimators=50, \
                                     max_depth=10, \
                                     min_samples_leaf=25, \
                                     max_features=2)
rf_model10.fit(X_train, y_train)
preds10 = rf_model10.predict(X_train)
test_preds10 = rf_model10.predict(X_test)
print(accuracy_score(y_train, preds10))
print(accuracy_score(y_test, test_preds10))
```

- The output will be as follows:

```
0.7810422474801629
0.7593062912705899
```

Maximum Features

```
rf_model11 = RandomForestClassifier(random_state=1, \
                                     n_estimators=50, \
                                     max_depth=10, \
                                     min_samples_leaf=25, \
                                     max_features=0.7)
rf_model11.fit(X_train, y_train)
preds11 = rf_model11.predict(X_train)
test_preds11 = rf_model11.predict(X_test)
print(accuracy_score(y_train, preds11))
print(accuracy_score(y_test, test_preds11))
```

- The output will be as follows:

```
0.792229608978483
0.7654016399390465
```

Maximum Features

```
rf_model12 = RandomForestClassifier(random_state=1, \
                                     n_estimators=50, \
                                     max_depth=10, \
                                     min_samples_leaf=25, \
                                     max_features='log2')
rf_model12.fit(X_train, y_train)
preds12 = rf_model12.predict(X_train)
test_preds12 = rf_model12.predict(X_test)
print(accuracy_score(y_train, preds12))
print(accuracy_score(y_test, test_preds12))
```

- The output will be as follows:

```
0.7810422474801629
0.7593062912705899
```

Complete Exercise 4.05: Tuning max_features

Complete Activity 4.01: Train a Random Forest Classifier on the ISOLET Dataset

Summary

- We have finally reached the end of this lesson on multiclass classification with Random Forest.
- We learned that multiclass classification is an extension of binary classification: instead of predicting only two classes, target variables can have many more values.
- We saw how we can train a Random Forest model in just a few lines of code and assess its performance by calculating the accuracy score for the training and testing sets



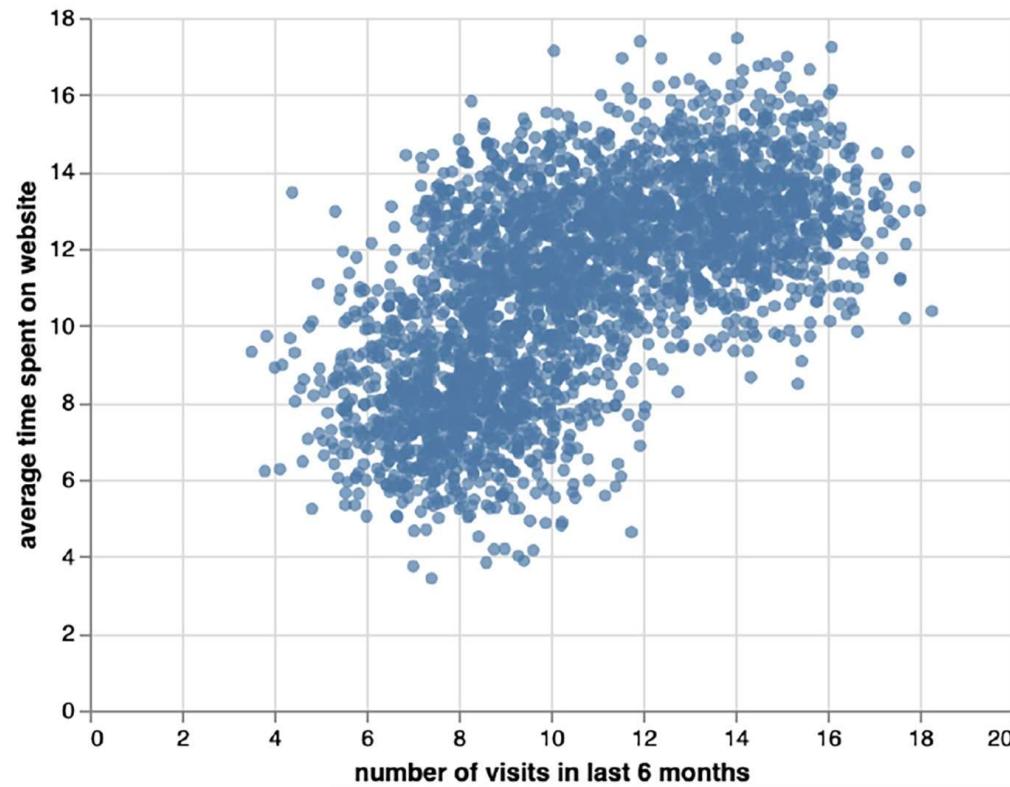
5. Performing Your First Cluster Analysis



Overview

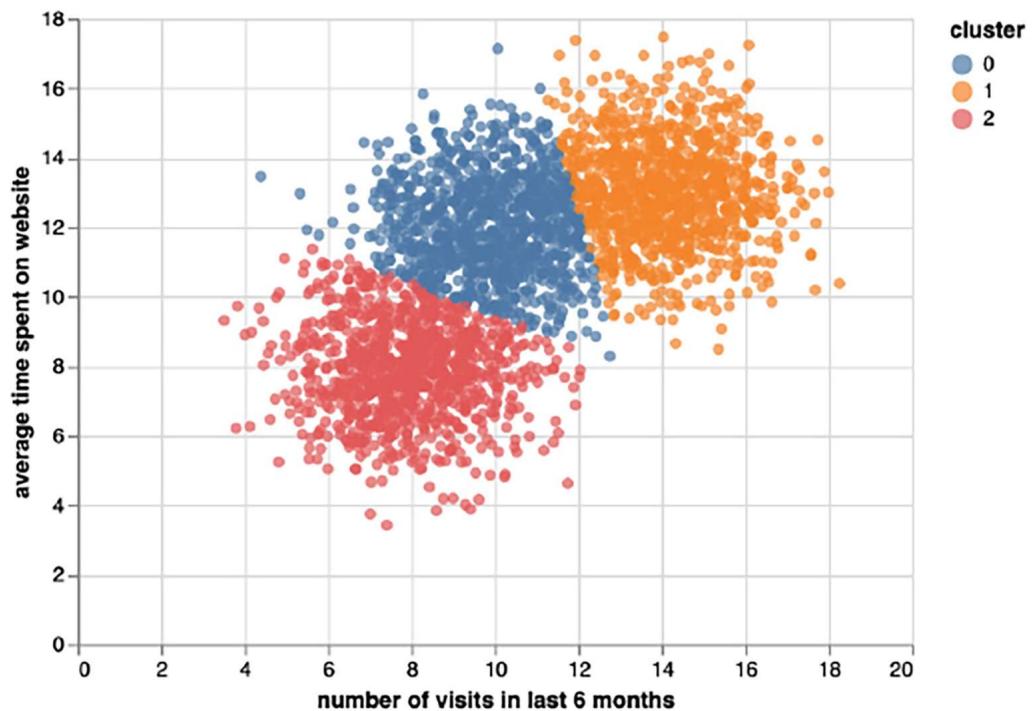
- This lesson will introduce you to unsupervised learning tasks, where algorithms have to automatically learn patterns from data by themselves as no target variables are defined beforehand.
- We will focus specifically on the k-means algorithm, and see how to standardize and process data for use in cluster analysis.

Introduction



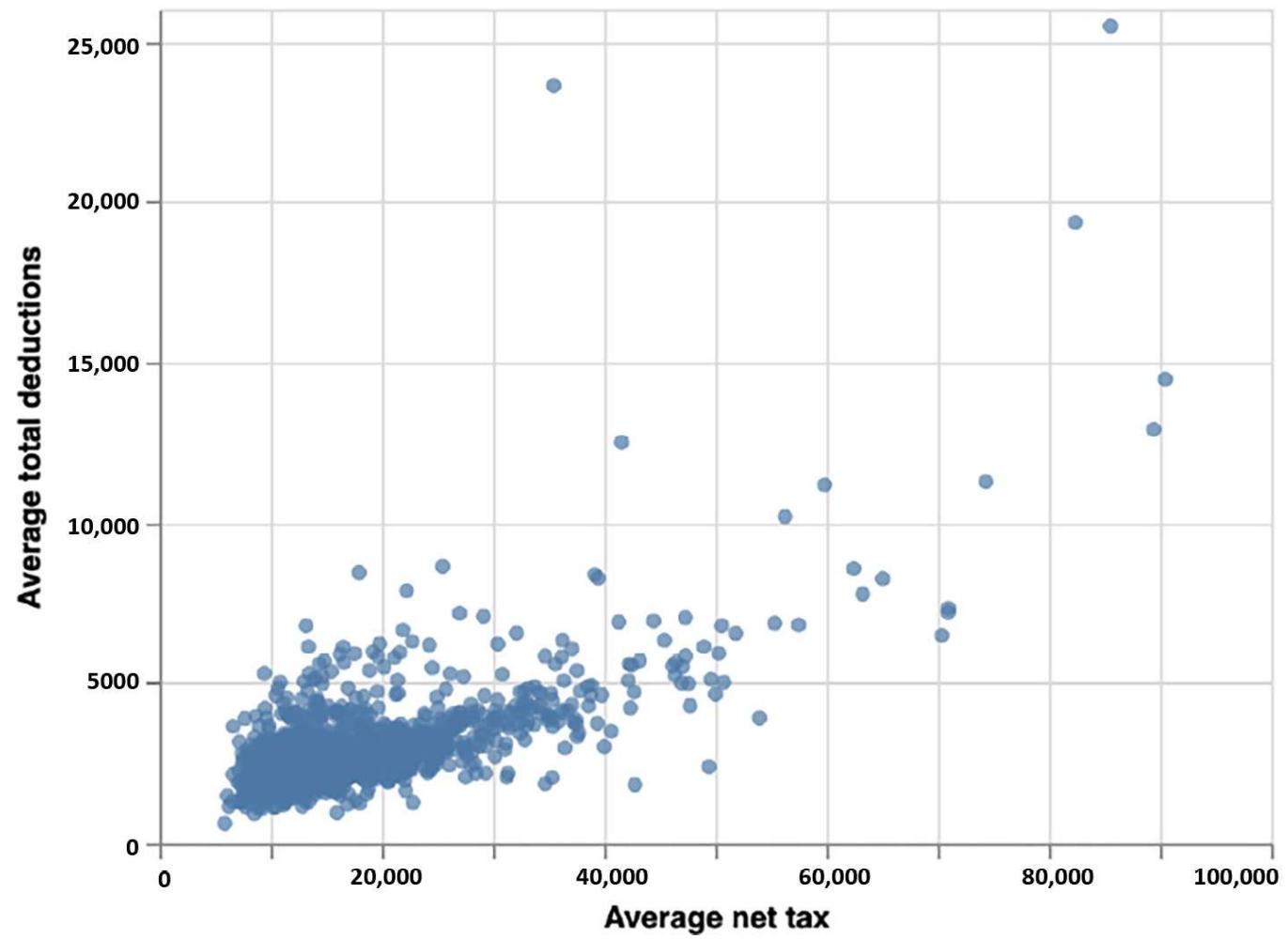
Introduction

- Clustering analysis performed on this data would uncover natural patterns by grouping similar data points such that you may get the following result:



Clustering with k-means

- k-means is one of the most popular clustering algorithms (if not the most popular) among data scientists due to its simplicity and high performance.
- Its origins date back as early as 1956, when a famous mathematician named Hugo Steinhaus laid its foundations
- but it was a decade later that another researcher called James MacQueen named this approach k-means.



Complete Exercise 5.01: Performing Your First Clustering Analysis on the ATO Dataset

Interpreting k-means Results

To create a pivot table similar to an Excel one, we will be using the `pivot_table()` method from pandas. We need to specify the following parameters for this method:

- `values`
- `Index`
- `aggfunc`

Interpreting k-means Results

- **aggfunc:** This is where you will specify the aggregation functions you want to summarize the data with, such as getting averages or counts.
- In Excel, this is the Summarize by option in the values field.
- An example of how to use the aggfunc method is shown below

```
import numpy as np  
df.pivot_table(values=['Average net tax', \  
                      'Average total deductions'], \  
                     index='cluster', aggfunc=np.mean)
```

cluster	Average net tax	Average total deductions
0	10126.249641	2176.045911
1	20474.671096	2909.352159
2	36197.407895	4789.328947
3	13016.528477	2480.054305
4	77733.111111	12523.111111
5	16391.342495	2657.048626
6	26765.963235	3498.676471
7	50887.692308	6156.961538

Interpreting k-means Results

- Let's import it first:

```
import altair as alt
```

- Then, we will instantiate a Chart() object with our DataFrame and save it into a variable called chart:

```
chart = alt.Chart(df)
```

Interpreting k-means Results

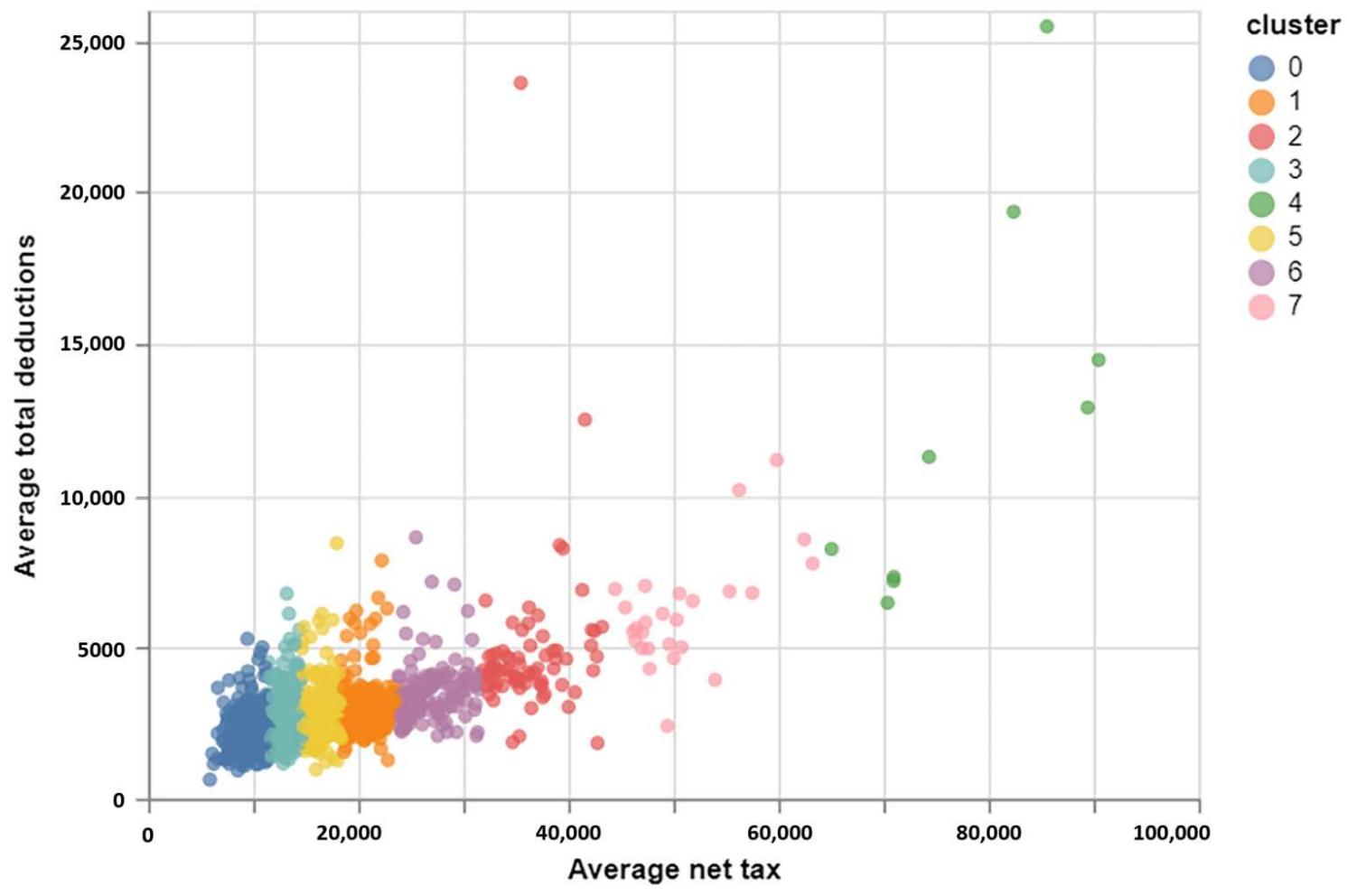
- Now we will specify the type of graph we want, a scatter plot, with the `.mark_circle()` method and will save it into a new variable called `scatter_plot`:

```
scatter_plot = chart.mark_circle()
```

Interpreting k-means Results

- Finally, we need to configure our scatter plot by specifying the names of the columns that will be our x- and y-axes on the graph.
- We also tell the scatter plot to color each point according to its cluster value with the color option:

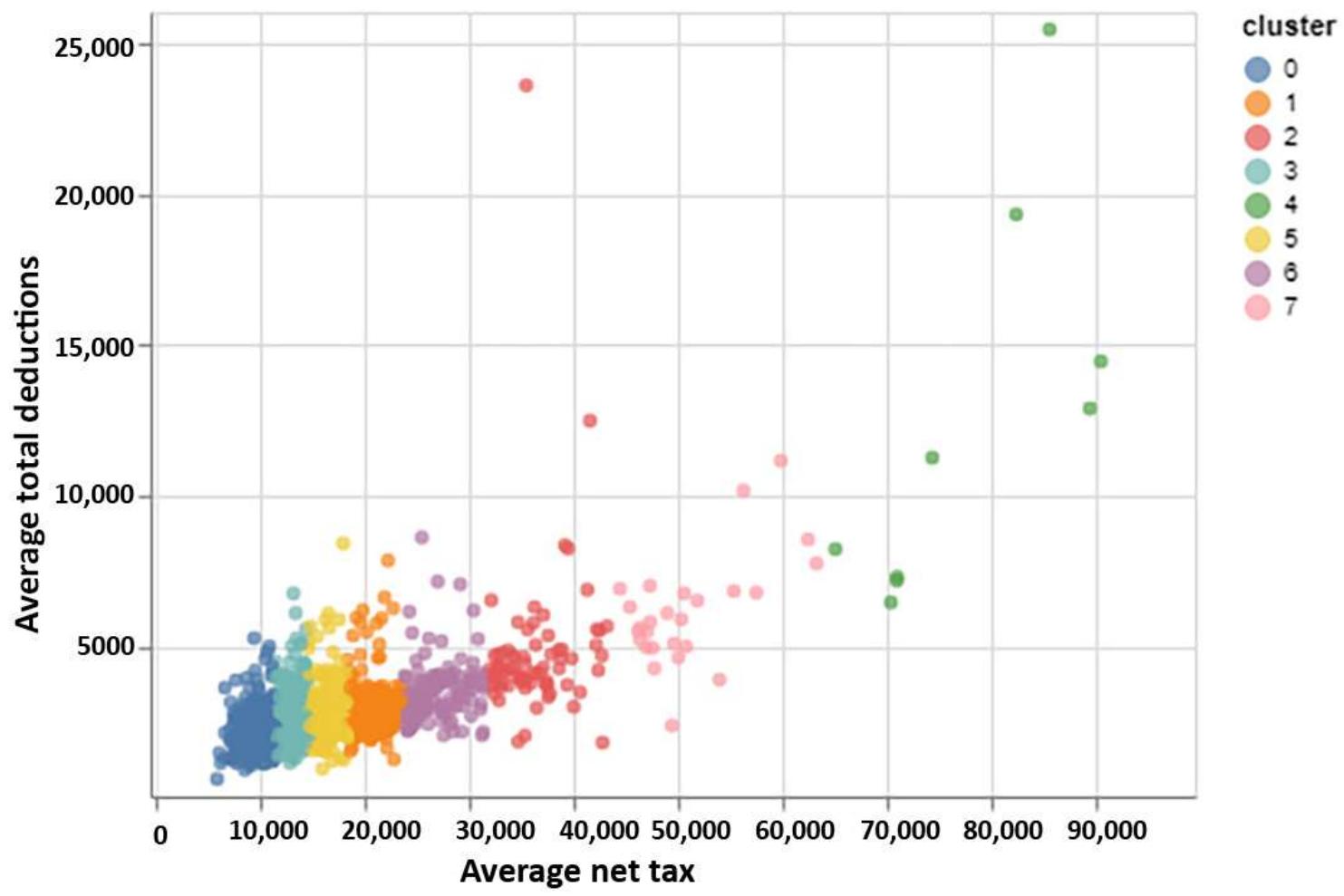
```
scatter_plot.encode(x='Average net tax', \
                    y='Average total deductions', \
                    color='cluster:N')
```



Interpreting k-means Results

- with a list of corresponding column names and call the `interactive()` method just after, as seen in the following code snippet:

```
scatter_plot.encode(x='Average net tax', \  
                    y='Average total deductions', \  
                    color='cluster:N', \  
                    tooltip=['Postcode', \  
                             'cluster', 'Average net tax', \  
                             'Average total deductions'])\ \  
.interactive()
```



Complete Exercise 5.02: Clustering Australian Postcodes by Business Income and Expenses

Choosing the Number of Clusters

- Let's apply this method to our ATO dataset. First, we will define the range of cluster numbers we want to evaluate (between 1 and 10) and save them in a DataFrame called clusters.
- We will also create an empty list called inertia, where we will store our calculated values.

```
clusters = pd.DataFrame()  
clusters['cluster_range'] = range(1, 10)  
inertia = []
```

Choosing the Number of Clusters

- Next, we will create a for loop that will iterate over the range, fit a k-means model with the specified number of clusters, extract the inertia value, and store it in our list, as in the following code snippet:

```
for k in clusters['cluster_range']:  
    kmeans = KMeans(n_clusters=k,  
random_state=8).fit(X)  
    inertia.append(kmeans.inertia_)
```

Choosing the Number of Clusters

- Now we can use our list of inertia values in the clusters DataFrame:
`clusters['inertia'] = inertia
clusters`

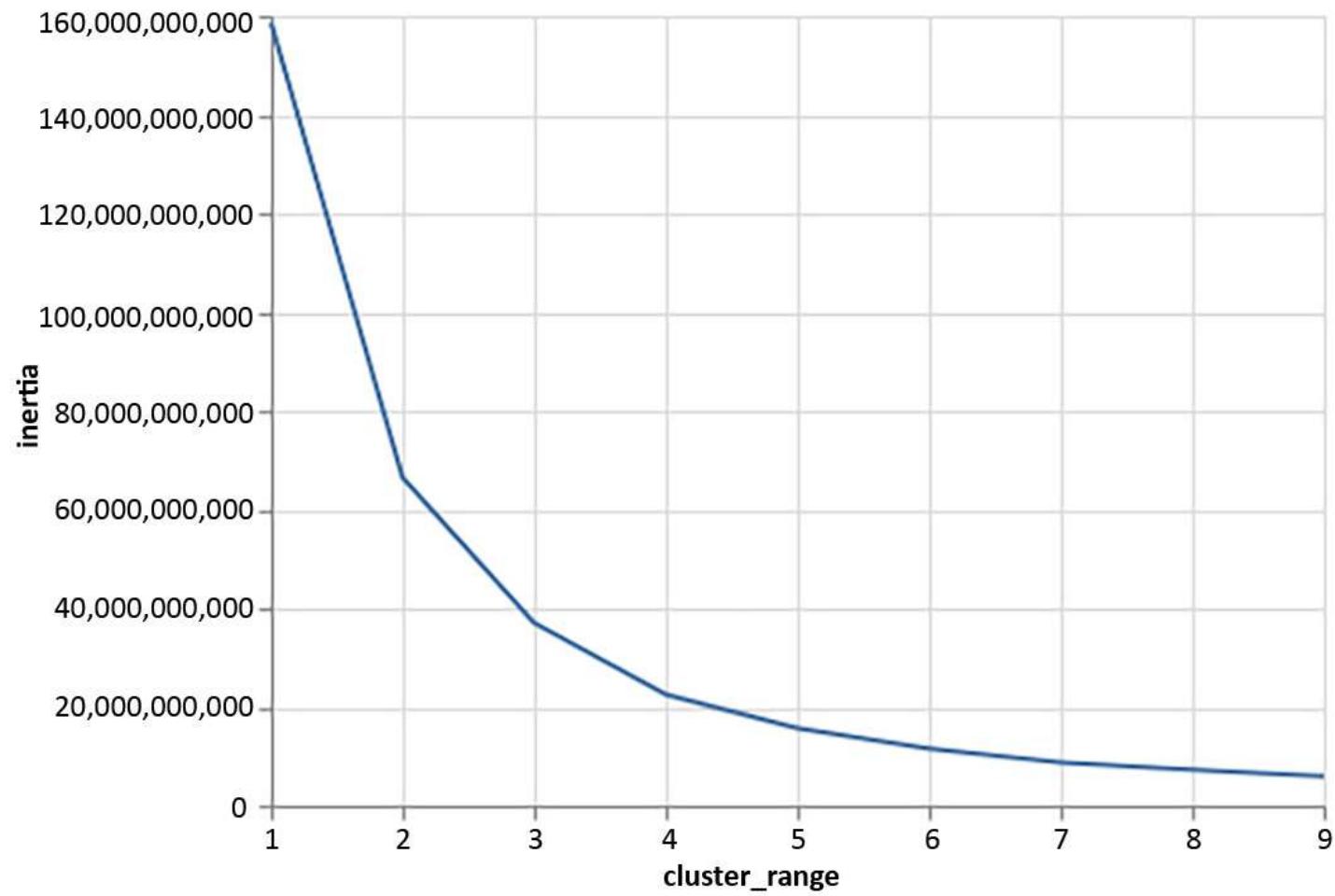
- You should get this output:

	cluster_range	inertia
0	1	1.586459e+11
1	2	6.655056e+10
2	3	3.708115e+10
3	4	2.253973e+10
4	5	1.575012e+10
5	6	1.154406e+10
6	7	8.776361e+09
7	8	7.285696e+09
8	9	6.000036e+09

Choosing the Number of Clusters

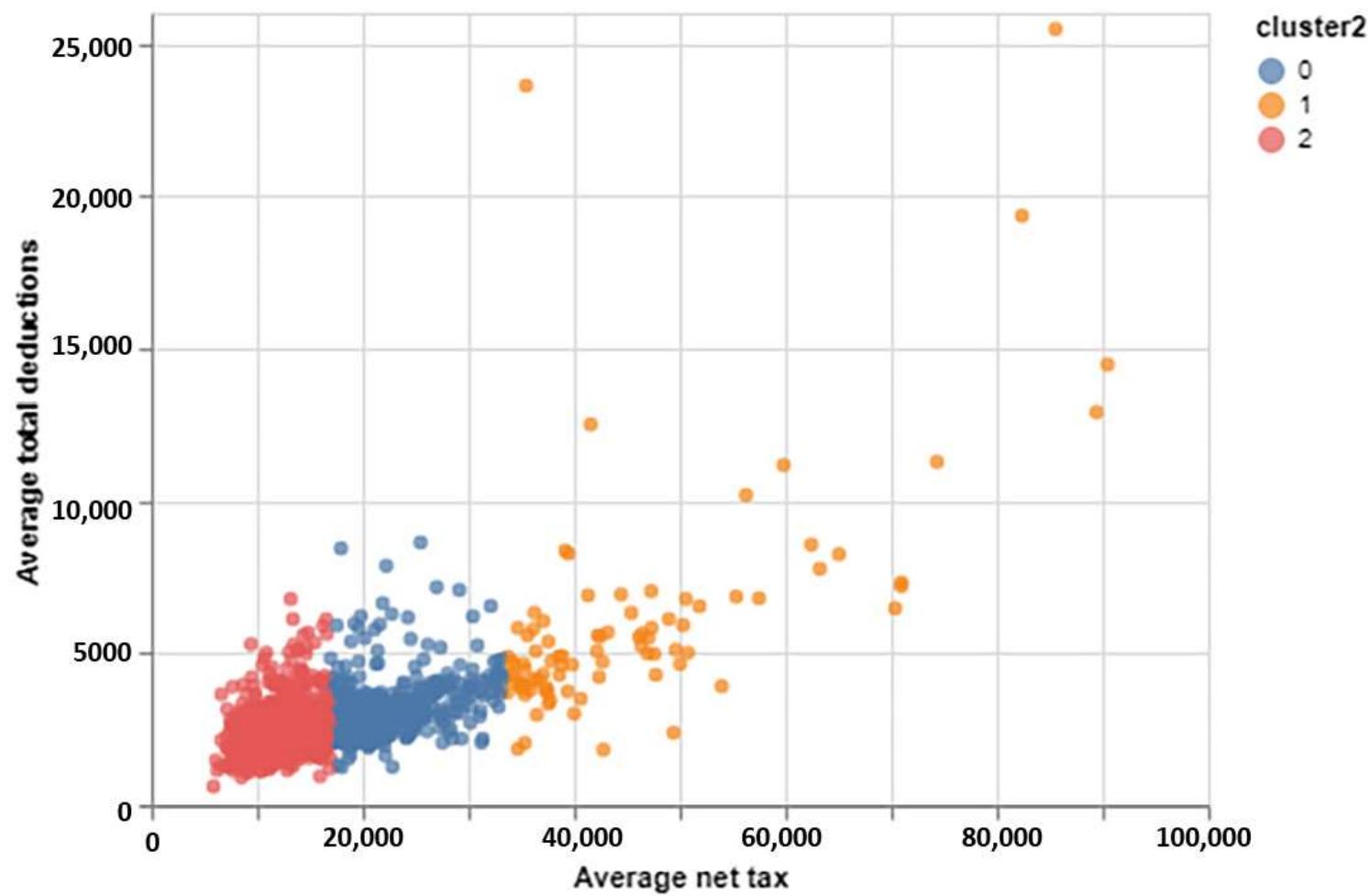
- Then, we need to plot a line chart using altair with the `mark_line()` method.
- We will specify the '`cluster_range`' column as our x-axis and '`inertia`' as our y-axis, as in the following code snippet:

```
alt.Chart(clusters).mark_line()\n    .encode(x='cluster_range', y='inertia')
```



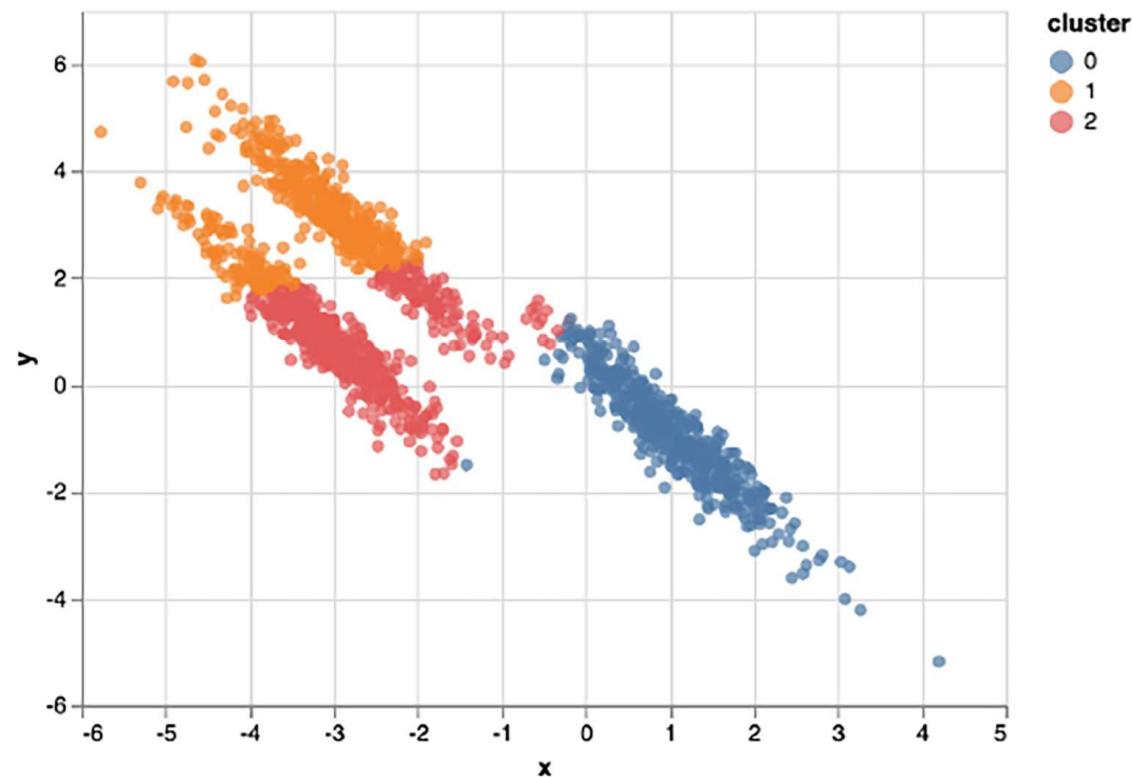
Choosing the Number of Clusters

```
kmeans = KMeans(random_state=42, n_clusters=3)
kmeans.fit(X)
df['cluster2'] = kmeans.predict(X)
scatter_plot.encode(x='Average net tax', \
                    y='Average total deductions', \
                    color='cluster2:N', \
                    tooltip=['Postcode', 'cluster', \
                            'Average net tax', \
                            'Average total deductions'])\
                    .interactive()
```



Complete Exercise 5.03: Finding the Optimal Number of Clusters

Initializing Clusters



Initializing Clusters

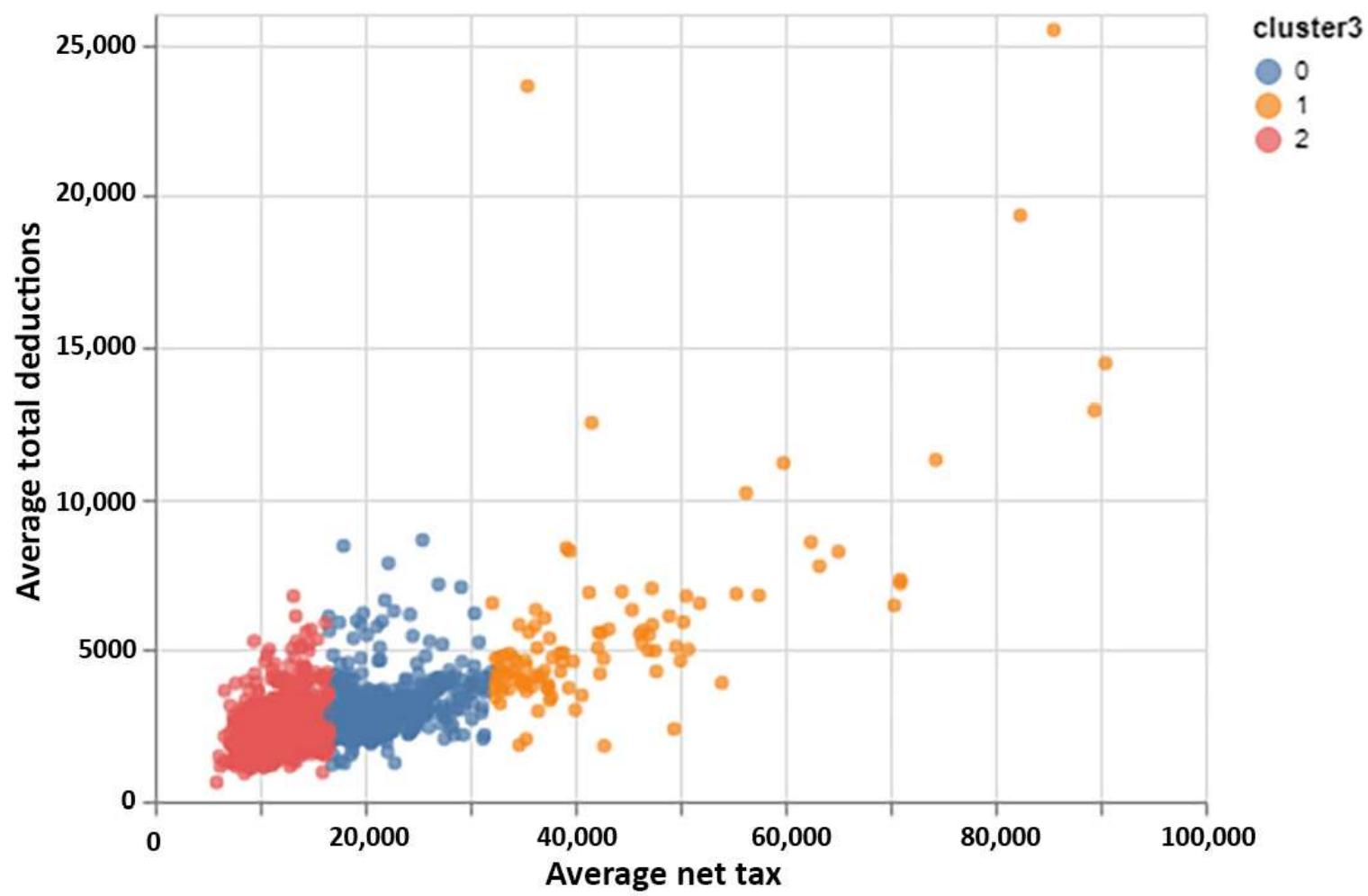
- First, let's run only one iteration using random initialization:

```
kmeans = KMeans(random_state=14, n_clusters=3, \
                  init='random', n_init=1)
kmeans.fit(X)
```

Initializing Clusters

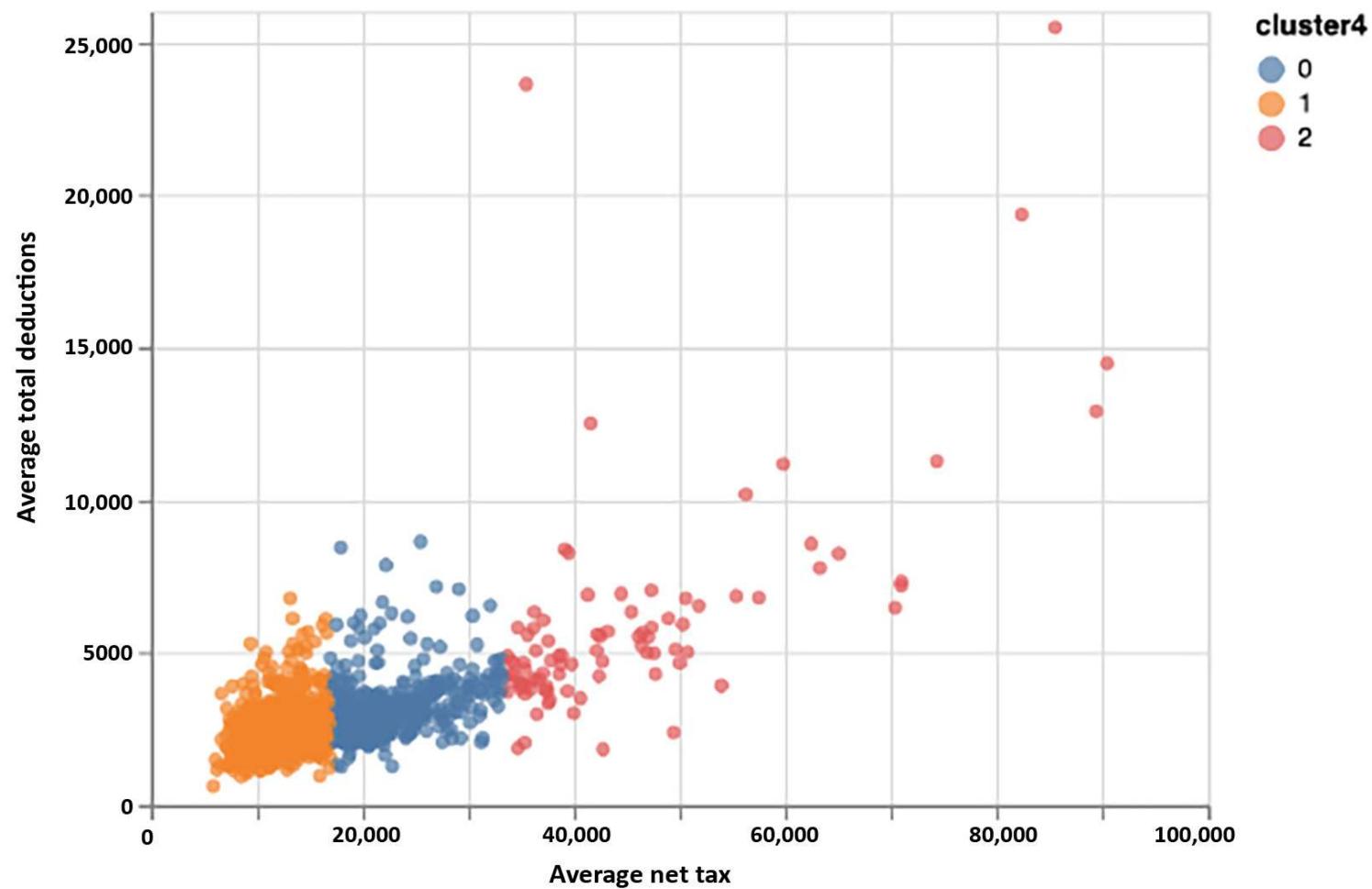
- As usual, we want to visualize our clusters with a scatter plot, as defined in the following code snippet:

```
df['cluster3'] = kmeans.predict(X)
alt.Chart(df).mark_circle()\
    .encode(x='Average net tax', \
            y='Average total deductions', \
            color='cluster3:N', \
            tooltip=['Postcode', 'cluster', \
                     'Average net tax', \
                     'Average total deductions']) \
    .interactive()
```



Initializing Clusters

```
kmeans = KMeans(random_state=14, n_clusters=3, \
                  init='k-means++', n_init=5)
kmeans.fit(X)
df['cluster4'] = kmeans.predict(X)
alt.Chart(df).mark_circle()\
    .encode(x='Average net tax', \
            y='Average total deductions', \
            color='cluster4:N', \
            tooltip=['Postcode', 'cluster', \
                     'Average net tax', \
                     'Average total deductions'])\
    .interactive()
```



Complete Exercise 5.04: Using Different Initialization Parameters to Achieve a Suitable Outcome

Calculating the Distance to the Centroid

- You are heading in the right direction.
- It has to do with some sort of distance measure between two points.
- The one used by k-means is called squared Euclidean distance and its formula is:

$$d(x, y)^2 = \sum_{i=1}^n (x_i - y_i)^2$$

Calculating the Distance to the Centroid

- Let's apply this formula to the ATO dataset.
- First, we will grab the values needed – that is, the coordinates from the first two observations – and print them:

```
x = X.iloc[0,:].values
```

```
y = X.iloc[1,:].values
```

```
print(x)
```

```
print(y)
```

- You should get the following output:

```
[27555 2071]
```

```
[28142 3804]
```

Calculating the Distance to the Centroid

- The coordinates for x are (27555, 2071) and the coordinates for y are (28142, 3804).
- Here, the formula is telling us to calculate the squared difference between each axis of the two data points and sum them:

```
squared_euclidean = (x[0] - y[0])**2 + (x[1] - y[1])**2  
print(squared_euclidean)
```

- You should get the following output:

3347858

Calculating the Distance to the Centroid

- First, we fit a k-means model as shown in the following code snippet:

```
kmeans = KMeans(random_state=42, n_clusters=3, \
                  init='k-means++', n_init=5)
kmeans.fit(X)
df['cluster6'] = kmeans.predict(X)
```

Calculating the Distance to the Centroid

- Now extract the centroids into a DataFrame and print them:

```
centroids = kmeans.cluster_centers_
centroids = pd.DataFrame(centroids, \
                        columns=['Average net tax', \
                                  'Average total deductions'])
print(centroids)
```

Calculating the Distance to the Centroid

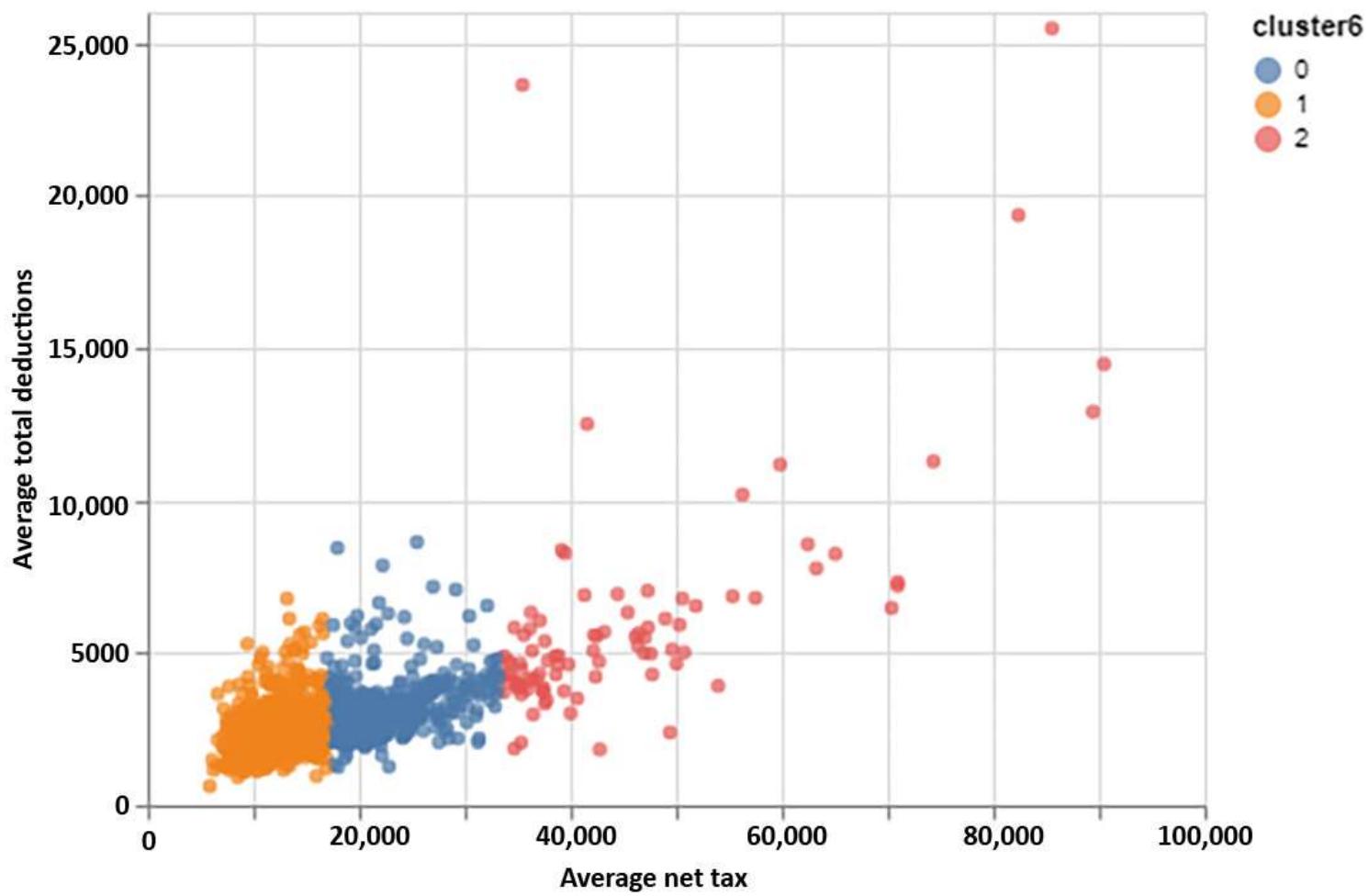
- You should get the following output:

	Average net tax	Average total deductions
0	21645.622924	3054.901993
1	12395.607303	2383.760674
2	45279.802198	6067.000000

Calculating the Distance to the Centroid

```
chart1 = alt.Chart(df).mark_circle()\n    .encode(x='Average net tax', \n            y='Average total deductions', \n            color='cluster6:N', \n            tooltip=['Postcode', 'cluster6', \n                     'Average net tax', \n                     'Average total deductions'])\n    .interactive()
```

chart1

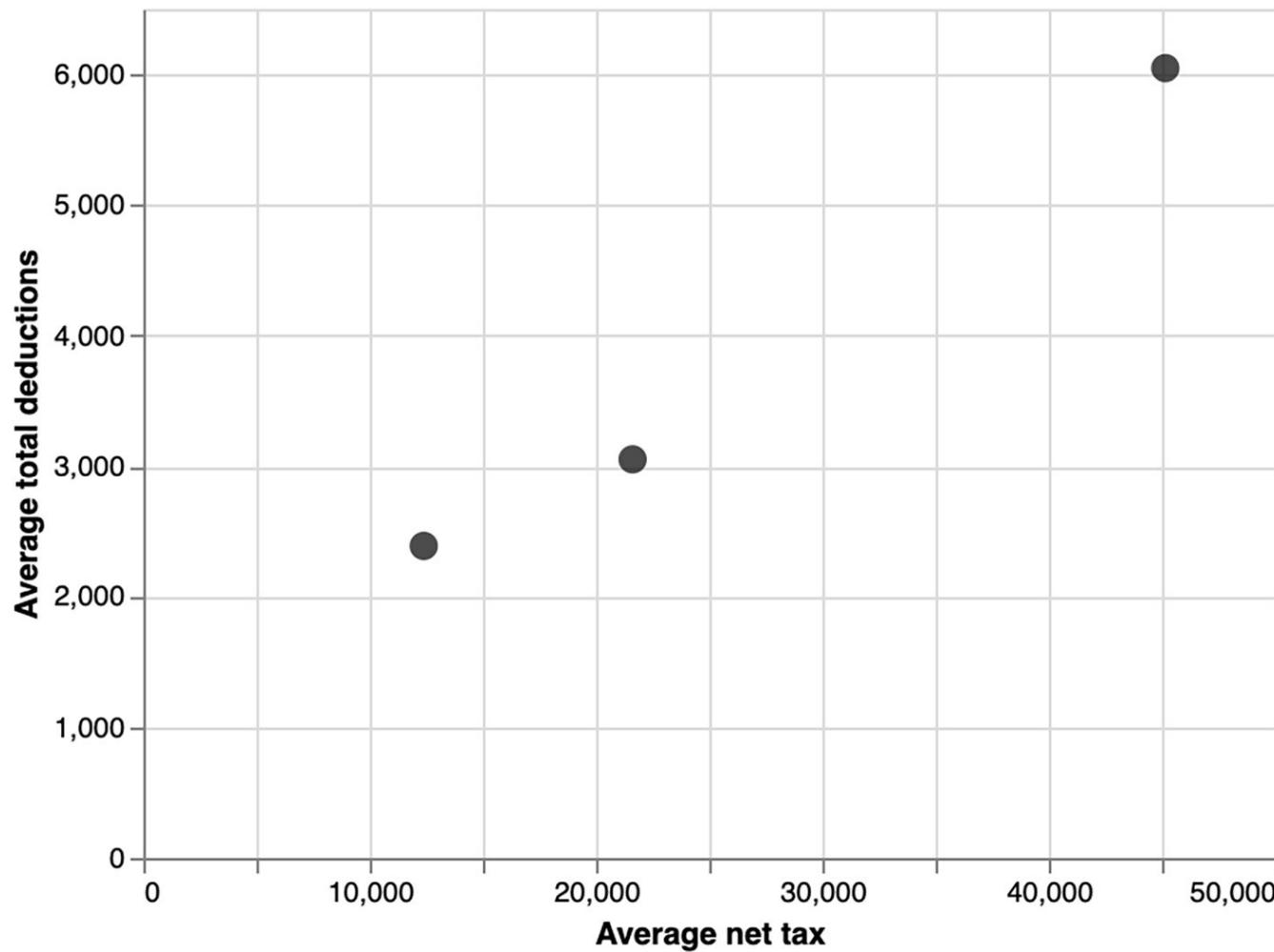


Calculating the Distance to the Centroid

- Now, to create a second scatter plot only for the centroids called chart2:

```
chart2 = alt.Chart(centroids).mark_circle(size=100)\n    .encode(x='Average net tax', \n            y='Average total deductions', \n            color=alt.value('black'), \n            tooltip=['Average net tax', \n                     'Average total deductions'])\n    .interactive()
```

chart2

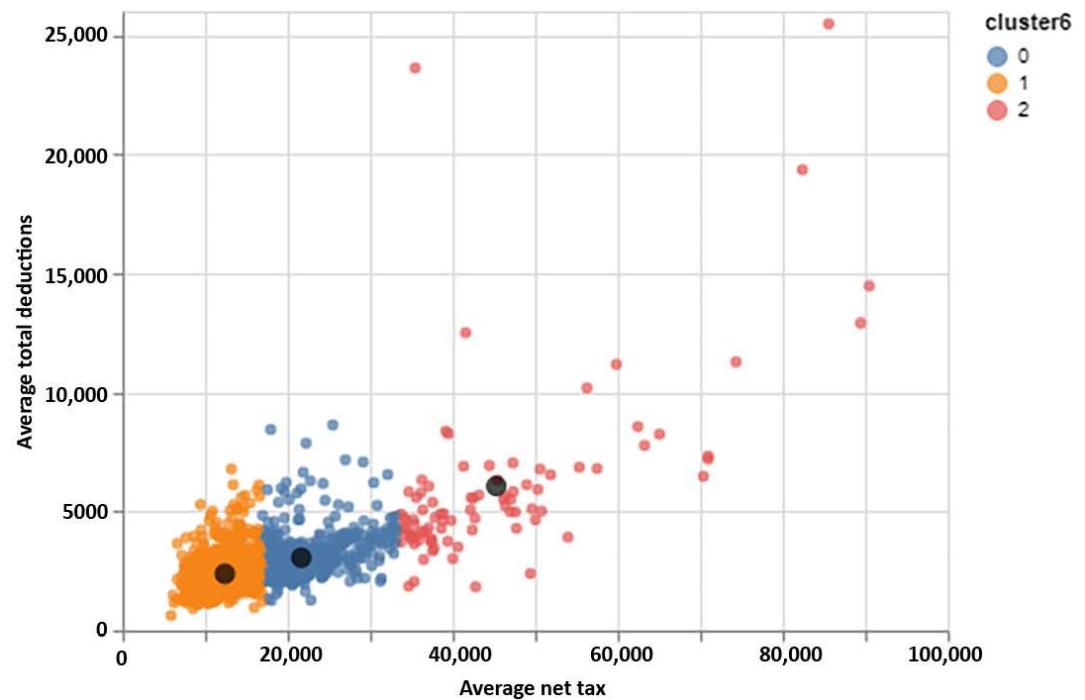


Calculating the Distance to the Centroid

- And now we combine the two charts, which is extremely easy with altair:

chart1 + chart2

- You should get this output:



Complete Exercise 5.05: Finding the Closest Centroids in Our Dataset

Standardizing Data

- Fitting a k-means algorithm is extremely easy.
- The trickiest part is making sure the resulting clusters are meaningful for your project, and we have seen how we can tune some hyperparameters to ensure this.
- But handling input data is as important as all the steps you have learned about so far.
- If your dataset is not well prepared, even if you find the best hyperparameters, you will still get some bad results.

Standardizing Data

- The formula for min-max scaling is very simple: on each axis, you need to remove the minimum value for each data point and divide the result by the difference between the maximum and minimum values.
- The scaled data will have values ranging between 0 and 1:

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

Standardizing Data

- First, we import the relevant class and instantiate an object:

```
from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler()
```

- Then, we fit it to our dataset:

```
min_max_scaler.fit(X)
```

- You should get the following output:

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

Standardizing Data

- And finally, call the transform() method to standardize the data:

```
X_min_max = min_max_scaler.transform(X)
```

```
X_min_max
```

- You should get the following output:

```
array([[0.25619932, 0.05804452],  
       [0.26313737, 0.1278026 ],  
       [0.11547644, 0.04472085],  
       ...,  
       [0.12640947, 0.06762468],  
       [0.27085549, 0.09700922],  
       [0.14493062, 0.06287485]])
```

Standardizing Data

- Now we print the minimum and maximum values of the min-max-scaled data for both axes:

```
X_min_max[:,0].min(), X_min_max[:,0].max(), \  
X_min_max[:,1].min(), X_min_max[:,1].max()
```

- You should get the following output:

```
(0.0, 1.0000000000000002, 0.0, 1.0)
```

Standardizing Data

- The z-score is calculated by removing the overall average from the data point and dividing the result by the standard deviation for each axis.
- The distribution of the standardized data will have a mean of 0 and a standard deviation of 1:

$$z_i = \frac{x_i - \bar{x}}{s}$$

Standardizing Data

- To apply it with sklearn, first, we have to import the relevant StandardScaler class and instantiate an object:

```
from sklearn.preprocessing import StandardScaler  
standard_scaler = StandardScaler()
```

- This time, instead of calling fit() and then transform(), we use the fit_transform() method:

```
X_scaled = standard_scaler.fit_transform(X)  
X_scaled
```

Standardizing Data

```
array([[ 1.47818338, -0.49490091],  
       [ 1.55236034,  0.90726658],  
       [-0.02633256, -0.76271247],  
       ...,  
       [ 0.09055617, -0.30233549],  
       [ 1.63487747,  0.28830632],  
       [ 0.288572   , -0.3978091 ]])
```

Standardizing Data

- Now we'll look at the minimum and maximum values for each axis:

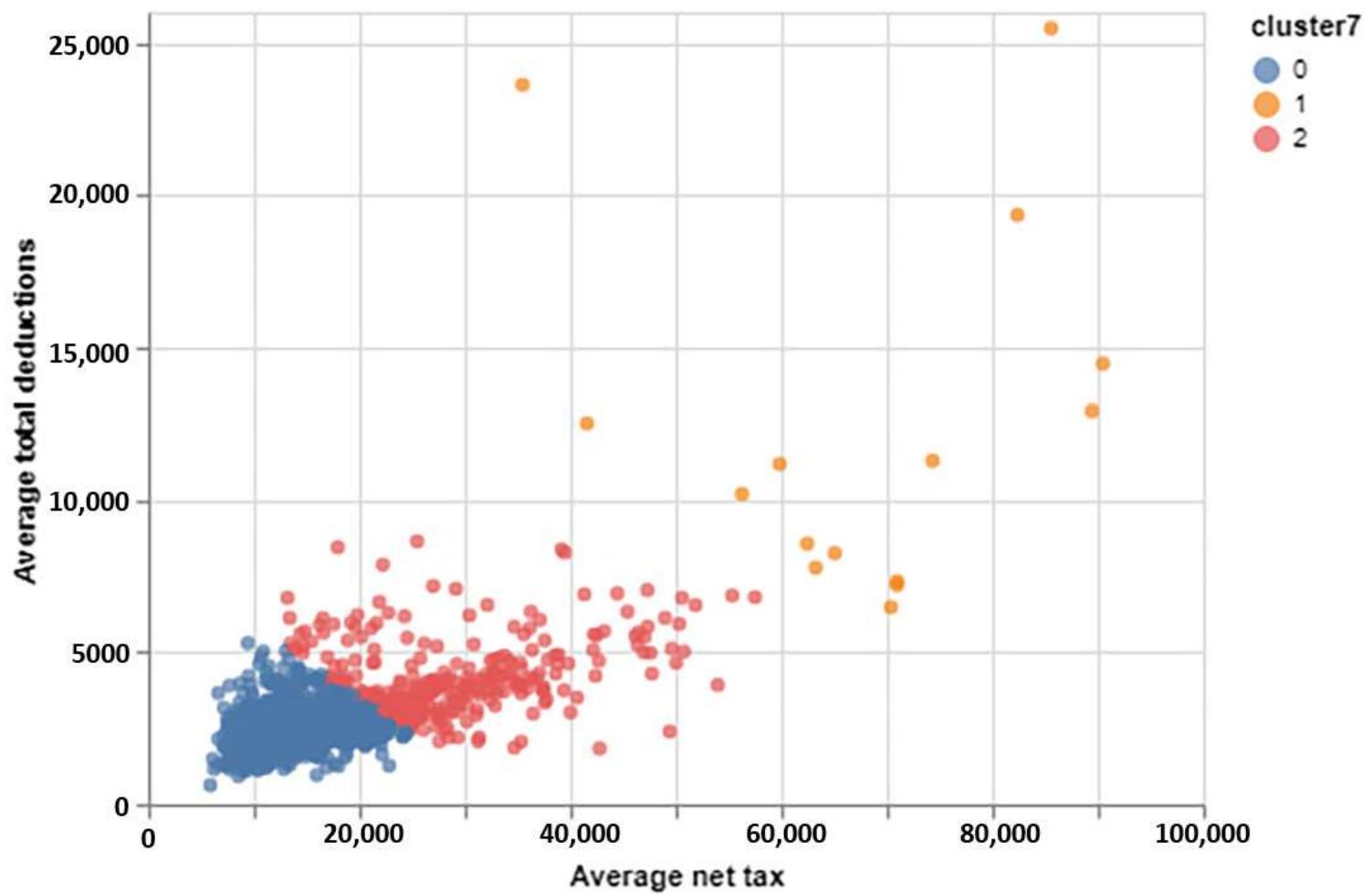
```
X_scaled[:,0].min(), X_scaled[:,0].max(), \
X_scaled[:,1].min(), X_scaled[:,1].max()
```

- You should get the following output:

```
(-1.2609303032180148,
 9.430408155273339,
 -1.6616207628956663,
 18.438810177350547)
```

Standardizing Data

```
kmeans = KMeans(random_state=42, n_clusters=3, \
                  init='k-means++', n_init=5)
kmeans.fit(X_scaled)
df['cluster7'] = kmeans.predict(X_scaled)
alt.Chart(df).mark_circle()\n    .encode(x='Average net tax', \
            y='Average total deductions', \
            color='cluster7:N', \
            tooltip=['Postcode', 'cluster7', \
                     'Average net tax', \
                     'Average total deductions'])\n    .interactive()
```



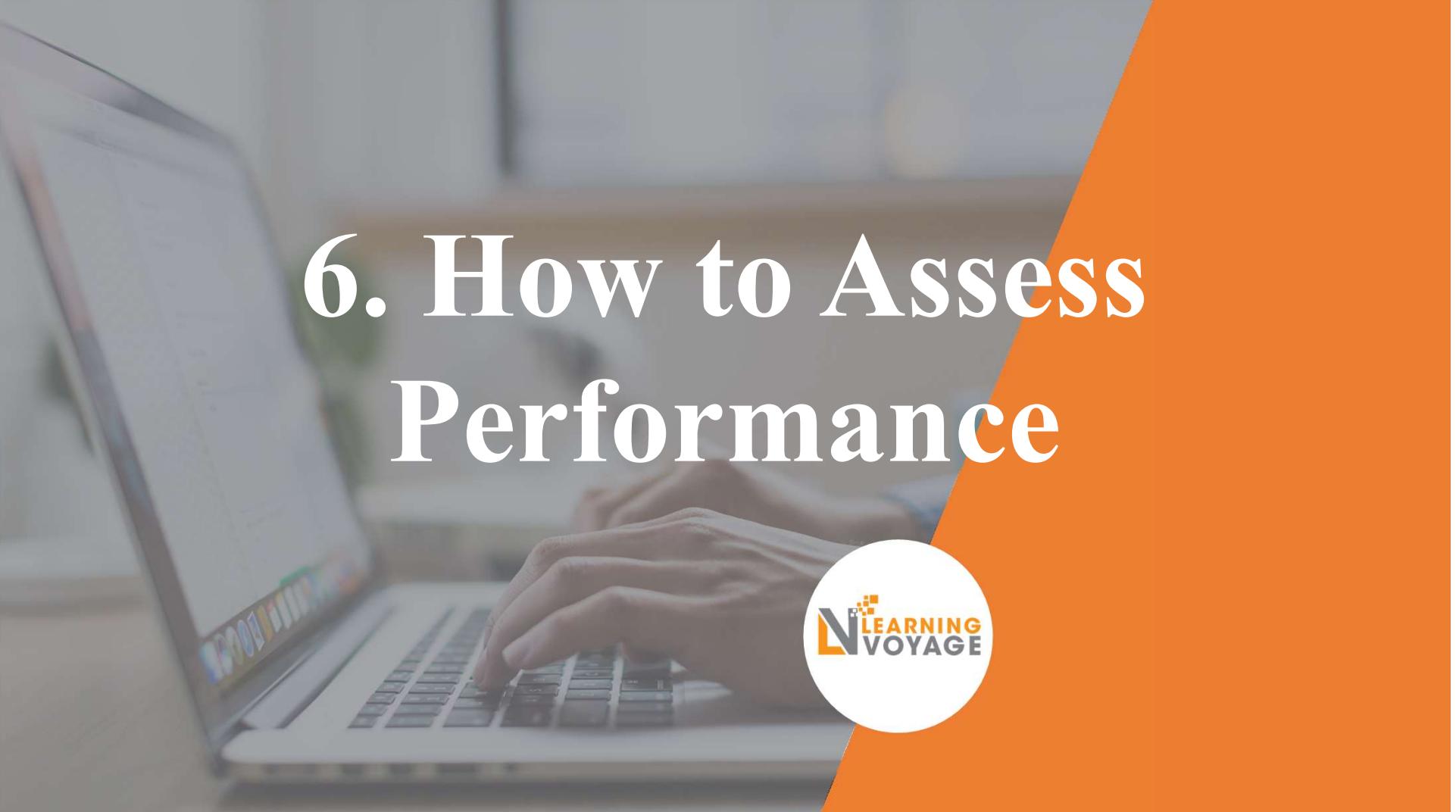
Complete Exercise 5.06: Standardizing the Data from Our Dataset

Complete Activity 5.01: Perform Customer Segmentation Analysis in a Bank Using k-means

Summary



- You are now ready to perform cluster analysis with the k-means algorithm on your own dataset.
- This type of analysis is very popular in the industry for segmenting customer profiles as well as detecting suspicious transactions or anomalies.
- We learned about a lot of different concepts, such as centroids and squared Euclidean distance.



6. How to Assess Performance



Overview

- This lesson will introduce you to model evaluation, where you evaluate or assess the performance of each model that you train before you decide to put it into production.
- By the end of this lesson, you will be able to create an evaluation dataset.

Introduction

- When you assess the performance of a model, you look at certain measurements or values that tell you how well the model is performing under certain conditions.
- That helps you make an informed decision about whether or not to make use of the model that you have trained in the real world.
- Some of the measurements you will encounter in this lesson are MAE, precision, recall, and R2 score

Splitting Data

You will learn more about splitting data in lesson 7, The Generalization of Machine Learning Models, where we will cover the following:

- Simple data splits using `train_test_split`
- Multiple data splits using cross-validation

Complete Exercise 6.01: Importing and Splitting Data

Assessing Model Performance for Regression Models

	actuals	predicted
0	4.891	4.132270
1	4.194	4.364320
2	4.984	4.440703
3	3.109	2.954363
4	5.115	4.987951

Scalars

- A scalar variable is a simple number, such as 23.
- Whenever you make use of numbers on their own, they are scalars.
- You assign them to variables, such as in the following expression:

temperature = 23

Scalars

- If you had to store the temperature for 5 days, you would need to store the values in 5 different values, such as in the following code snippet:

```
temp_1 = 23  
temp_2 = 24  
temp_3 = 23  
temp_4 = 22  
temp_5 = 22
```

Vectors

- Vectors look similar to Python lists and can be created from Python lists.
- Consider the following code snippet for creating a Python list:

```
temps_list = [23, 24, 23, 22, 22]
```

Vectors

- You can create a vector from the list using the `.array()` method from numpy by first importing numpy and then using the following snippet:

```
import numpy as np  
temps_ndarray = np.array(temps_list)
```

Vectors

- You can proceed to verify the data type using the following code snippet:
`print(type(temp_ndarray))`
- The code snippet will cause the compiler to print out the following:

```
<class 'numpy.ndarray'>
```

Vectors

- You may inspect the contents of the vector using the following code snippet:

```
print(temp_ndarray)
```

- This generates the following output:

```
[ 23  24  23  22  22 ]
```

Vectors

- Note that the output contains single square brackets, [and], and the numbers are separated by spaces.
- This is different from the output from a Python list, which you can obtain using the following code snippet:

```
print(temp_list)
```

- The code snippet yields the following output:

```
[23, 24, 23, 22, 22]
```

Vectors

- Vectors have a shape and a dimension. Both of these can be determined by using the following code snippet:

```
print(temp_ndarray.shape)
```

- The output is a Python data structure called a tuple and looks like this:

(5,)

Matrices

- A matrix is also made up of scalars but is different from a scalar in the sense that a matrix has both rows and columns
- To convert it into a matrix with five rows and one column, you would use the following snippet:

```
temps_matrix = temps_ndarray.reshape(-1, 1)
```

Matrices

- The second parameter, 1, instructs the interpreter to add a new dimension.
- This new dimension is the column.
- To see the new shape, use the following snippet:
`print(temp_matrix.shape)`

- You will get the following output:

(5, 1)

Matrices

- Notice that the tuple now has two numbers, 5 and 1.
- The first number, 5, represents the rows, and the second number, 1, represents the columns.
- You can print out the value of the matrix using the following snippet:
`print(temp_matrix)`
- The output of the code is as follows:

```
[[23]
 [24]
 [23]
 [22]
 [22]]
```

Matrices

- You may reshape the matrix to contain 1 row and 5 columns and print out the value using the following code snippet:

```
print(temp_matrix.reshape(1,5))
```

- The output will be as follows:

```
[ [ 23  24  23  22  22 ] ]
```

Matrices

- Finally, you can convert the matrix back into a vector by dropping the column using the following snippet:

```
vector = temps_matrix.reshape(-1)
```

- You can print out the value of the vector to confirm that you get the following:

```
[ 23  24  23  22  22 ]
```

R2 Score

- The R2 score (pronounced "r squared") is sometimes called the "score" and measures the coefficient of determination of the model.
- Think of it as the model's ability to make good and reliable predictions.
- This measure is accessed using the `score()` method of the model and is available for every model.

Complete Exercise 6.02: Computing the R² Score of a Linear Regression Model

Mean Absolute Error

- The mean absolute error (MAE) is an evaluation metric for regression models that measures the absolute distance between your predictions and the ground truth.
- The absolute distance is the distance regardless of the sign, whether positive or negative.
- For example, if the ground truth is 6 and you predict 5, the distance is 1.

Complete Exercise 6.03: Computing the MAE of a Model

Complete Exercise 6.04: Computing the Mean Absolute Error of a Second Model

Other Evaluation Metrics

- While we made use of `mean_absolute_error`, there are other model evaluation functions for regression.
- Recall that these are all cost (or loss) functions.
- These include `max_error`, `mean_squared_error`, `mean_squared_log_error`, and `median_absolute_error`.

Assessing Model Performance for Classification Models

- Classification models are used for predicting which class a group of features will fall under.
- You learned to create binary classification models in lesson 3, Binary Classification, and multi-class classification models in lesson 4, Multiclass Classification with RandomForest.

Complete Exercise 6.05: Creating a Classification Model for Computing Evaluation Metrics

The Confusion Matrix

- You encountered the confusion matrix in lesson 3, Binary Classification.
- You may recall that the confusion matrix compares the number of classes that the model predicted against the actual occurrences of those classes in the validation dataset.
- The output is a square matrix that has the number of rows and columns equal to the number of classes you are predicting.

Complete Exercise 6.06: Generating a Confusion Matrix for the Classification Model

More on the Confusion Matrix

- The confusion matrix helps you analyze the impact of the choices you would have to make if you put the model into production.
- Let's consider the example of predicting the presence of a disease based on the inputs to the model.
- This is a binary classification problem, where 1 implies that the disease is present and 0 implies the disease is absent.
- The confusion matrix for this model would have two columns and two rows.

Precision

- Suppose 10 entries were classified as positive.
- If 7 of the entries were actually positive, then TP would be 7 and FP would be 3.
- The precision would, therefore, be 0.7. The equation is given as follows:

$$\frac{tp}{tp + fp}$$

Complete Exercise 6.07: Computing Precision for the Classification Model

Recall

- Recall is the total number of predictions that were true divided by the number of predictions for the class, both true and false.
- Think of it as the true positive divided by the sum of entries in the column.
- The equation is given as follows:

$$\frac{tp}{tp + fn}$$

Complete Exercise 6.08: Computing Recall for the Classification Model

F1 Score

- The F1 score is another important parameter that helps us to evaluate the model performance.
- It considers the contribution of both precision and recall using the following equation:

$$\frac{2 * \textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}}$$

Complete Exercise 6.09: Computing the F1 Score for the Classification Model

Accuracy

- Accuracy is an evaluation metric that is applied to classification models.
- It is computed by counting the number of labels that were correctly predicted, meaning that the predicted label is exactly the same as the ground truth.
- The accuracy_score() function exists in sklearn.metrics to provide this value.

Complete Exercise 6.10: Computing Model Accuracy for the Classification Model

Logarithmic Loss

- The logarithmic loss (or log loss) is the loss function for categorical models.
- It is also called categorical cross-entropy. It seeks to penalize incorrect predictions.
- The sklearn documentation defines it as "the negative log-likelihood of the true values given your model predictions."

Complete Exercise 6.11: Computing the Log Loss for the Classification Model

Receiver Operating Characteristic Curve

- Recall the True Positive Rate, which we discussed earlier, It is also called sensitivity.
- Also recall that what we try to do with a logistic regression model is find a threshold value such that above that threshold value
- We predict that our input falls into a certain class, and below that threshold, we predict that it doesn't.

Complete Exercise 6.12: Computing and Plotting ROC Curve for a Binary Classification Problem

Area Under the ROC Curve

- The Area Under the Receiver Operating Characteristic Curve (ROC AUC) is a measure of the likelihood that the model will rank a randomly chosen positive example higher than a randomly chosen negative example.
- Another way of putting it is to say that the higher this measure is, the better the model is at predicting a negative class as negative, and a positive class as positive.

Complete Exercise 6.13: Computing the ROC AUC for the Caesarian Dataset

Saving and Loading Models

- You will eventually need to transfer some of the models you have trained to a different computer so they can be put into production.
- There are various utilities for doing this, but the one we will discuss is called joblib.

Complete Exercise 6.14: Saving and Loading a Model

**Complete Activity 6.01: Train Three Different Models
and Use Evaluation Metrics to Pick the Best
Performing Model**

Summary



- In this lesson we observed that some of the evaluation metrics for classification models require a binary classification model.
- We saw that when we worked with more than two classes, we were required to use the one-versus-all approach.
- The one-versus-all approach builds one model for each class and tries to predict the probability that the input belongs to a specific class.

7. The Generalization of Machine Learning Models



Overview

- This lesson will teach you how to make use of the data you have to train better models by either splitting your data if it is sufficient or making use of cross-validation if it is not.
- By the end of this lesson, you will know how to split your data into training, validation, and test datasets.
- You will be able to identify the ratio in which data has to be split and also consider certain features while splitting.

Introduction

Generalization deals with getting your models to perform well enough on data points that they have not encountered in the past (that is, during training). We will address two specific areas:

- How to make use of as much of your data as possible to train a model
- How to reduce overfitting in a model

Overfitting

Overfitting occurs when a model has been over-engineered. Two of the ways in which this could occur are:

- The model is trained on too many features.
- The model is trained for too long.

Training on Too Many Features

- When a model trains on too many features, the hypothesis becomes extremely complicated.
- Consider a case in which you have one column of features and you need to generate a hypothesis.
- This would be a simple linear equation, as shown here:

$$y = w \cdot x_1$$

Training on Too Many Features

- Now, consider a case in which you have two columns, and in which you cross the columns by multiplying them.
- The hypothesis becomes the following:

$$y = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_1 \cdot x_2 + b$$

Training on Too Many Features

- While the first equation yields a line, the second equation yields a curve, because it is now a quadratic equation.
- But the same two features could become even more complicated depending on how you engineer your features.
- Consider the following equation:

$$y = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_1 \cdot x_2 + w_4 \cdot x_1^2 + w_5 \cdot x_2^3 + b$$

Training for Too Long

- The model starts training by initializing the vector of weights such that all values are equal to zero.
- During training, the weights are updated according to the gradient update rule.
- This systematically adds or subtracts a small value to each weight.
- As training progresses, the magnitude of the weights increases. If the model trains for too long, these model weights become too large.

Underfitting

- Consider an alternative situation in which the data has 10 features, but you only make use of 1 feature.
- Your model hypothesis would still be the following:

$$y = w \cdot x_1$$

Data

- A training dataset, which is used to train your model and measure the training loss.
- An evaluation or validation dataset, which you use to measure the validation loss of the model to see whether the validation loss continues to reduce as well as the training loss.
- A test dataset for final testing to see how well the model performs before you put it into production.

The Ratio for Dataset Splits

- The evaluation dataset is set aside from your entire training data and is never used for training.
- There are various schools of thought around the particular ratio that is set aside for evaluation, but it generally ranges from a high of 30% to a low of 10%.
- This evaluation dataset is normally further split into a validation dataset that is used during training and a test dataset that is used at the end for a sanity check.

Creating Dataset Splits

- Pick items whose indices correspond to the random numbers previously generated.

items	23	18	19	65	80	55	7	29	66	95
indices	0	1	2	3	4	5	6	7	8	9
Random : Numbers	2					↑ 5			↑ 8	

Complete Exercise 7.01: Importing and Splitting Data

Random State

- Consider the following figure as an example, The columns are your random states.
- If you pick 0 as the random state, the following numbers will be generated: 41, 52, 38, 56...
- However, if you pick 1 as the random state, a different set of numbers will be generated, and so on.

0	1	2	3	4	5	6	7	8	9	10
41	12	77	91	5	85	48	61	83	41	32
52	41	84	37	43	99	34	19	67	94	18
38	76	84	73	88	36	90	12	39	86	94
56	7	36	88	32	35	90	30	19	14	2
69	26	15	74	60	43	85	49	82	7	41
28	28	7	0	75	72	29	40	82	9	87
25	35	19	36	35	88	9	55	33	80	41
37	83	2	29	6	20	31	75	98	96	16
72	50	52	7	50	67	9	1	45	12	0
95	20	21	80	35	2	48	68	39	62	95
1	86	5	54	74	4	44	85	86	73	64
86	16	70	23	20	45	1	95	83	98	49
96	49	26	57	88	35	59	26	40	19	97
20	3	41	21	81	99	19	93	14	62	7
97	96	87	92	82	71	6	54	69	86	16
3	42	41	70	5	59	28	90	41	74	73
51	9	3	23	14	7	95	10	75	9	12
19	42	33	71	51	11	51	50	87	28	45
85	72	57	36	91	9	86	19	54	23	32
40	45	54	17	39	49	58	18	3	54	53
80	62	48	7	86	4	58	69	72	10	22
28	21	44	97	28	14	82	37	77	12	10
24	35	81	63	64	20	78	98	2	69	8
56	49	30	0	91	12	93	8	75	69	55
36	22	25	95	67	66	27	53	11	51	36
33	58	4	91	78	25	27	20	61	15	80
45	94	22	14	67	25	55	9	92	74	52
72	0	4	53	27	40	42	37	95	76	60
70	44	99	89	81	50	65	39	14	0	52
76	99	54	95	22	37	40	84	43	14	58

Complete Exercise 7.02: Setting a Random State When Splitting Data

Cross-Validation

- Consider an example where you split your data into five parts of 20% each.

Split data
into 5 parts

A	B	C	D	E
---	---	---	---	---

- Validation = A; Training = B, C, D, E
- Validation = B; Training = A, C, D, E
- Validation = C; Training = A, B, D, E
- Validation = D; Training = A, B, C, E
- Validation = E; Training = A, B, C, D

KFold

- The KFold class in `sklearn.model_selection` returns a generator that provides a tuple with two indices, one for training and another for testing or validation.
- A generator function lets you declare a function that behaves like an iterator, thus letting you use it in a loop.

Complete Exercise 7.03: Creating a Five-Fold Cross-Validation Dataset

Complete Exercise 7.04: Creating a Five-Fold Cross-Validation Dataset Using a Loop for Calls

cross_val_score

- The `cross_val_score()` function is available in `sklearn.model_selection`.
- Up until this point, you have learned how to create cross-validation datasets in a loop.
- If you made use of that approach, you would need to keep track of all of the models that you are training and evaluating inside of that loop.

Complete Exercise 7.05: Getting the Scores from Five-Fold Cross-Validation

Understanding Estimators That Implement CV

- Split the data using something like Kfold().
- Iterate over the number of splits and create an estimator.
- Train and evaluate each estimator.
- Pick the estimator with the best metrics to use, You have already seen various approaches to doing that.

LogisticRegressionCV

- LogisticRegressionCV is a class that implements cross-validation inside it.
- This class will train multiple LogisticRegression models and return the best one.

Complete Exercise 7.06: Training a Logistic Regression Model Using Cross-Validation

Hyperparameter Tuning with GridSearchCV

- GridSearchCV will take a model and parameters and train one model for each permutation of the parameters.
- At the end of the training, it will provide access to the parameters and the model scores.
- This is called hyperparameter tuning and you will be looking at this in much more depth in lesson 8, Hyperparameter Tuning.

Decision Trees

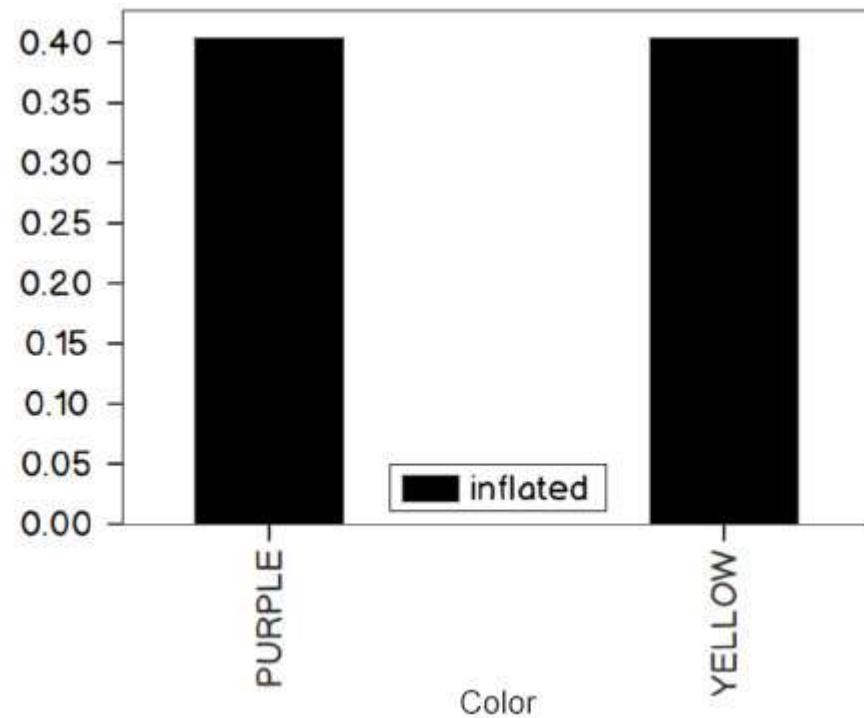
Consider the following data about balloons. The label you need to predict is called inflated. This dataset is used for predicting whether the balloon is inflated or deflated given the features. The features are:

- color
- size
- act
- age

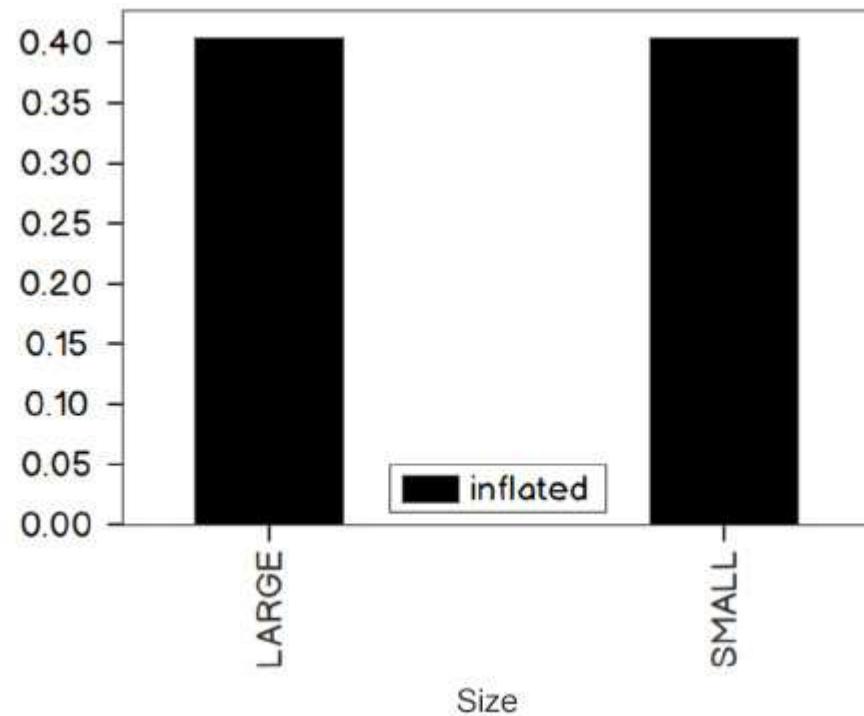
Decision Trees

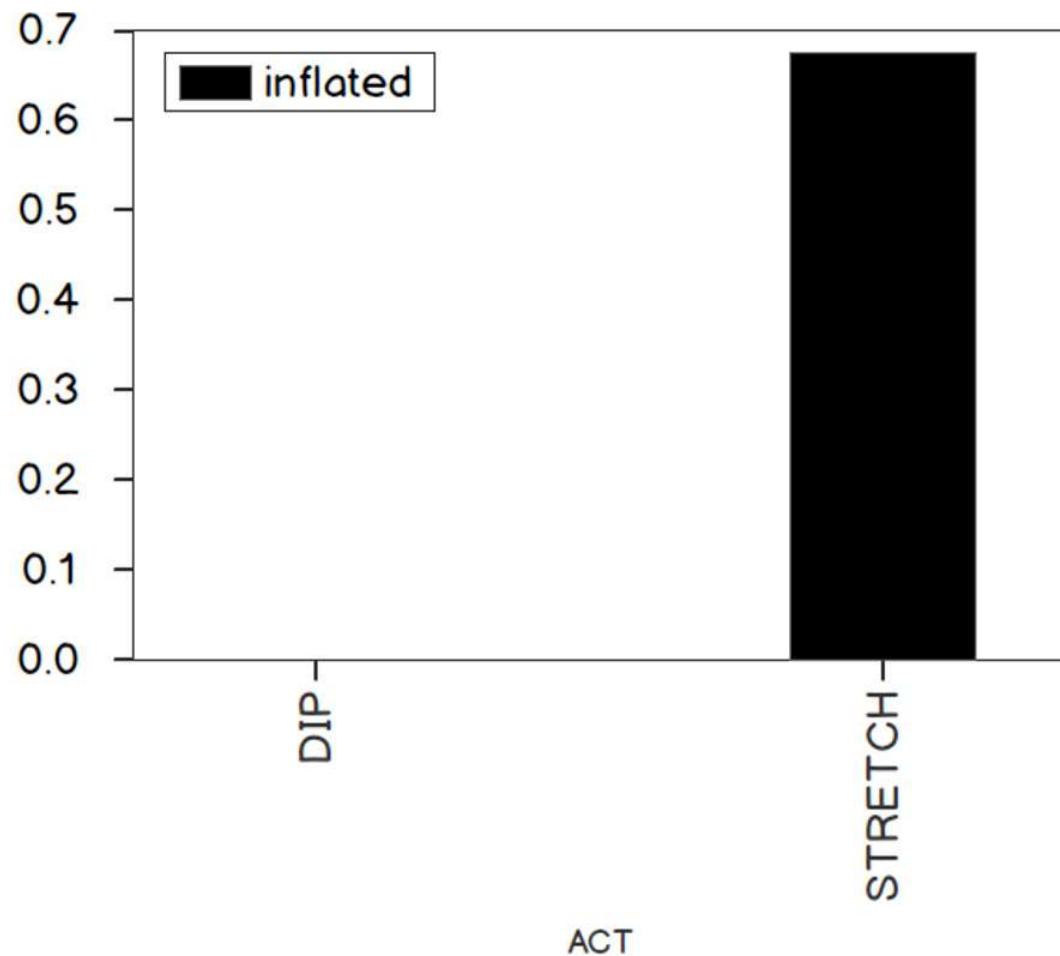
	color	size	act	age	inflated
0	YELLOW	SMALL	STRETCH	ADULT	T
1	YELLOW	SMALL	STRETCH	ADULT	T
2	YELLOW	SMALL	STRETCH	CHILD	F
3	YELLOW	SMALL	DIP	ADULT	F
4	YELLOW	SMALL	DIP	CHILD	F
5	YELLOW	LARGE	STRETCH	ADULT	T
6	YELLOW	LARGE	STRETCH	ADULT	T
7	YELLOW	LARGE	STRETCH	CHILD	F
8	YELLOW	LARGE	DIP	ADULT	F
9	YELLOW	LARGE	DIP	CHILD	F
10	PURPLE	SMALL	STRETCH	ADULT	T
11	PURPLE	SMALL	STRETCH	ADULT	T
12	PURPLE	SMALL	STRETCH	CHILD	F
13	PURPLE	SMALL	DIP	ADULT	F
14	PURPLE	SMALL	DIP	CHILD	F
15	PURPLE	LARGE	STRETCH	ADULT	T
16	PURPLE	LARGE	STRETCH	ADULT	T
17	PURPLE	LARGE	STRETCH	CHILD	F
18	PURPLE	LARGE	DIP	ADULT	F
19	PURPLE	LARGE	DIP	CHILD	F

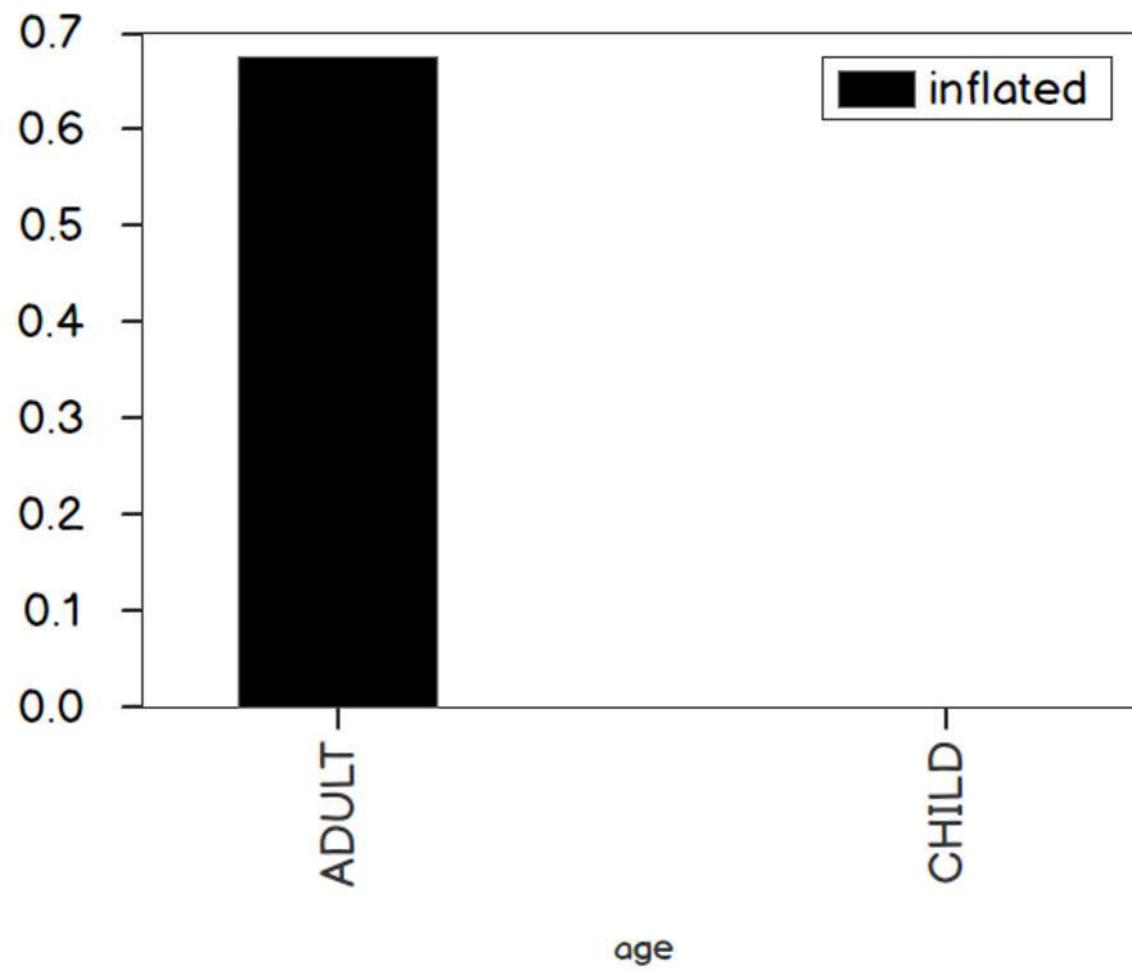
Decision Trees



Decision Trees

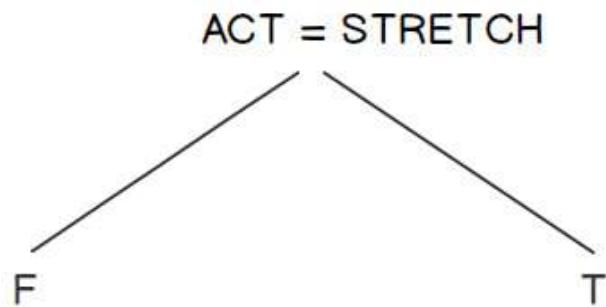




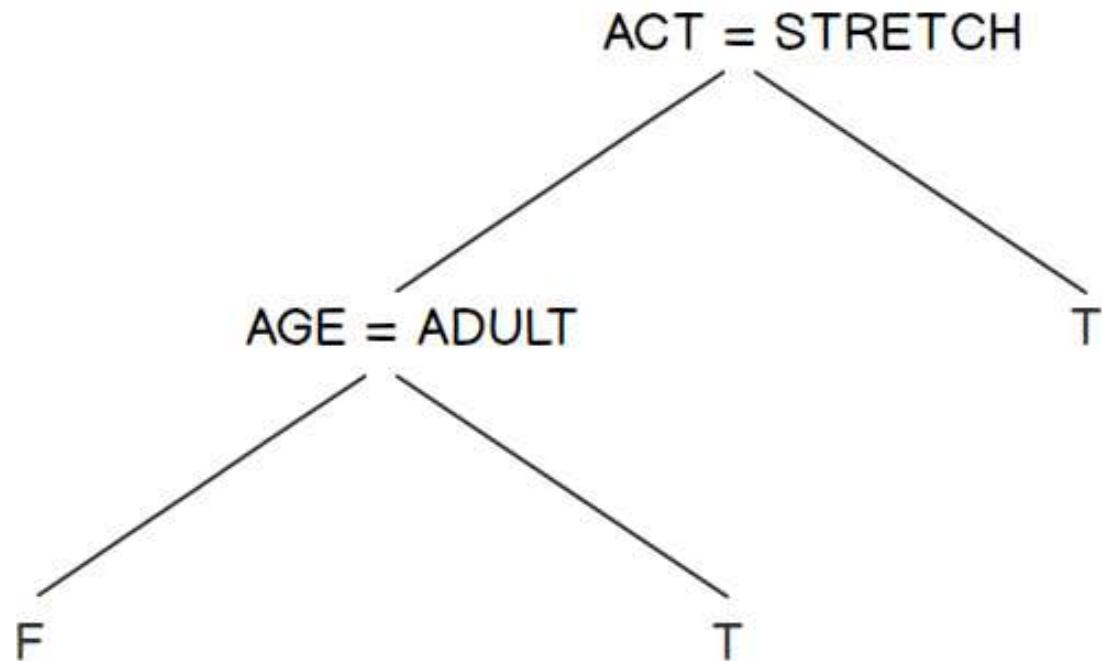


Decision Trees

- The two features that are useful to the decision tree are Act and Age.
- The tree could start by considering whether Act is STRETCH.
- If it is, the prediction will be true.
- This tree would look like the following figure:



Decision Trees



Complete Exercise 7.07: Using Grid Search with Cross-Validation to Find the Best Parameters for a Model

Hyperparameter Tuning with RandomizedSearchCV

- Grid search goes over the entire search space and trains a model or estimator for every combination of parameters.
- Randomized search goes over only some of the combinations.
- This is a more optimal use of resources and still provides the benefits of hyperparameter tuning and cross-validation.

Complete Exercise 7.08: Using Randomized Search for Hyperparameter Tuning

Model Regularization with Lasso Regression

You may recall that weights are optimized during model training. One method for optimizing weights is called gradient descent. The gradient update rule makes use of a differentiable loss function. Examples of differentiable loss functions are:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)

Complete Exercise 7.09: Fixing Model Overfitting Using Lasso Regression

Ridge Regression

- You just learned about lasso regression, which introduces a penalty and tries to eliminate certain features from the data.
- Ridge regression takes an alternative approach by introducing a penalty that penalizes large weights.

Complete Exercise 7.10: Fixing Model Overfitting Using Ridge Regression

Complete Activity 7.01: Find an Optimal Model for Predicting the Critical Temperatures of Superconductors

Summary

- In this lesson, we studied the importance of withholding some of the available data to evaluate models.
- We also learned how to make use of all of the available data with a technique called cross-validation to find the best performing model from a set of models you are training.



8. Hyperparameter Tuning



Overview

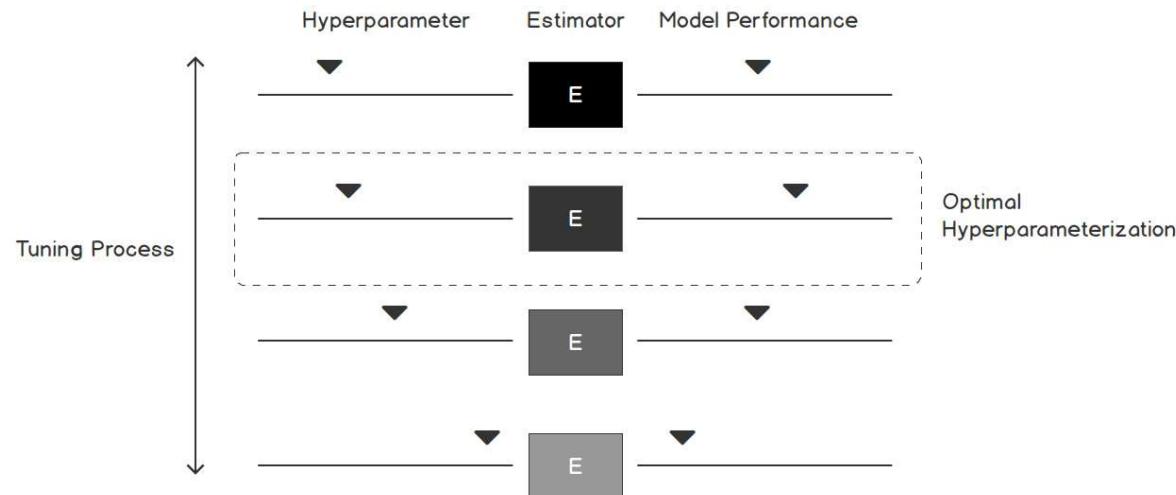
- In this lesson, each hyperparameter tuning strategy will be first broken down into its key steps before any high-level scikit-learn implementations are demonstrated.
- This is to ensure that you fully understand the concept behind each of the strategies before jumping to the more automated methods.

Introduction

- Depending on which estimator you eventually select, there may be settings that can be adjusted to improve overall predictive performance.
- These settings are known as hyperparameters, and deriving the best hyperparameters is known as tuning or optimizing.
- Properly tuning your hyperparameters can result in performance improvements well into the double-digit percentages, so it is well worth doing in any modeling exercise.

What Are Hyperparameters?

- Hyperparameters can be thought of as a set of dials and switches for each estimator that change how the estimator works to explain relationships in the data.



Difference between Hyperparameters and Statistical Model Parameters

- These terms relate to the parameters that feature in the mathematical formulation of models.
- The simplest example is that of the single variable linear model with no intercept term that takes the following form:

$$y = \beta X$$

Setting Hyperparameters

- For instance, say we initialize a k-NN estimator without specifying any arguments (empty brackets).
- To see the default hyperparameterization, we can run:

```
from sklearn import neighbors
# initialize with default hyperparameters
knn = neighbors.KNeighborsClassifier()
# examine the defaults
print(knn.get_params())
```

Setting Hyperparameters

- You should get the following output:

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',  
'metric_params': None, 'n_jobs': None, 'n_neighbors': 5,  
'p': 2, 'weights': 'uniform'}
```

Setting Hyperparameters

- To get more information as to what these parameters mean, how they can be changed, and what their likely effect may be, you can run the following command and view the help file for the estimator in question.
- For our k-NN estimator:

?knn

```
Type:          KNeighborsClassifier
String form:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
File:         /usr/local/lib/python3.6/dist-packages/sklearn/neighbors/classification.py
Docstring:
Classifier implementing the k-nearest neighbors vote.

Read more in the :ref:`User Guide <classification>`.

Parameters
-----
n_neighbors : int, optional (default = 5)
    Number of neighbors to use by default for :meth:`kneighbors` queries.

weights : str or callable, optional (default = 'uniform')
    weight function used in prediction. Possible values:

    - 'uniform' : uniform weights. All points in each neighborhood
      are weighted equally.
    - 'distance' : weight points by the inverse of their distance.
      in this case, closer neighbors of a query point will have a
      greater influence than neighbors which are further away.
    - [callable] : a user-defined function which accepts an
      array of distances, and returns an array of the same shape
      containing the weights.
```

Setting Hyperparameters

- Coming back to our example, if we want to change the hyperparameterization from $k = 5$ to $k = 15$, just re-initialize the estimator and set the `n_neighbors` argument to 15, which will override the default:

initialize with $k = 15$ and all other hyperparameters as default

```
knn = neighbors.KNeighborsClassifier(n_neighbors=15)
# examine
print(knn.get_params())
```

Setting Hyperparameters

- You should get the following output:

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',  
 'metric_params': None, 'n_jobs': None, 'n_neighbors':  
 15,  
 'p': 2, 'weights': 'uniform'}
```

Setting Hyperparameters

- For more information, see the description for the weights argument in the help file (?knn):

"""

initialize with k = 15, weights = distance and all other hyperparameters as default

"""

```
knn = neighbors.KNeighborsClassifier(n_neighbors=15, \
weights='distance')
```

```
# examine
```

```
print(knn.get_params())
```

Setting Hyperparameters

- The output will be as follows:

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski',  
 'metric_params': None, 'n_jobs': None, 'n_neighbors': 15,  
 'p': 2, 'weights': 'distance'}
```

Finding the Best Hyperparameterization

- The best hyperparameterization depends on your overall objective in building a machine learning model in the first place.
- In most cases, this is to find the model that has the highest predictive performance on unseen data, as measured by its ability to correctly label data points (classification) or predict a number (regression).

Complete Exercise 8.01: Manual Hyperparameter Tuning for a k-NN Classifier

Advantages and Disadvantages of a Manual Search

- Of all the strategies for hyperparameter tuning, the manual process gives you the most control.
- As you go through the process, you can get a feel for how your estimators might perform under different hyperparameterizations, and this means you can adjust them in line with your expectations without having to try a large number of possibilities unnecessarily.

Tuning Using Grid Search

- In the context of machine learning, grid search refers to a strategy of systematically testing out every hyperparameterization from a pre-defined set of possibilities for your chosen estimator.
- You decide the criteria used to evaluate performance, and once the search is complete, you may manually examine the results and choose the best hyperparameterization, or let your computer automatically choose it for you.

Simple Demonstration of the Grid Search Strategy

- In the following demonstration of the grid search strategy, we will use the breast cancer prediction dataset we saw in Exercise, Manual Hyperparameter Tuning for a k-NN Classifier, where we manually tuned the hyperparameters of the k-NN classifier to optimize for the precision of cancer predictions.

Simple Demonstration of the Grid Search Strategy

- The code will be as follows:

```
from sklearn import neighbors, datasets, model_selection  
# load data  
cancer = datasets.load_breast_cancer()  
# target  
y = cancer.target  
# features  
X = cancer.data  
# hyperparameter grid  
grid = {'k': [1, 3, 5, 7]}
```

Simple Demonstration of the Grid Search Strategy

```
# for every value of k in the grid
for k in grid['k']:
    # initialize the knn estimator
    knn = neighbors.KNeighborsClassifier(n_neighbors=k)
    # conduct a 10-fold cross-validation
    cv = model_selection.cross_val_score(knn, X, y, cv=10, \
                                          scoring='precision')
    # calculate the average precision value over all folds
    cv_mean = round(cv.mean(), 3)
    # report the result
    print('With k = {}, mean precision = {}'.format(k, cv_mean))
```

Simple Demonstration of the Grid Search Strategy

- The output will be as follows:

With $k = 1$, mean precision = 0.919

With $k = 3$, mean precision = 0.928

With $k = 5$, mean precision = 0.936

With $k = 7$, mean precision = 0.931

Simple Demonstration of the Grid Search Strategy

- To make this point clear, we can run the same loop, this time just printing the hyperparameterization that will be tried:

```
# for every value of k in the grid
for k in grid['k']:
    # initialize the knn estimator
    knn =
        neighbors.KNeighborsClassifier(n_neighbors=k)
    # print the hyperparameterization
    print(knn.get_params())
```

Simple Demonstration of the Grid Search Strategy

- The output will be as follows:

Refer to the file 8_1.txt

- To proceed, all we need to do is create a dictionary containing both the values of k and the weight functions we would like to try as separate key/value pairs:

Refer to the file 8_2.txt

Simple Demonstration of the Grid Search Strategy

With k = 1 and weight function = uniform, mean precision = 0.919

With k = 1 and weight function = distance, mean precision = 0.919

With k = 3 and weight function = uniform, mean precision = 0.928

With k = 3 and weight function = distance, mean precision = 0.929

With k = 5 and weight function = uniform, mean precision = 0.936

With k = 5 and weight function = distance, mean precision = 0.93

With k = 7 and weight function = uniform, mean precision = 0.931

With k = 7 and weight function = distance, mean precision = 0.926

Simple Demonstration of the Grid Search Strategy

```
# for every value of k in the grid
for k in grid['k']:
    # and every possible weight_function in the grid
    for weight_function in grid['weight_function']:
        # initialize the knn estimator
        knn = neighbors.KNeighborsClassifier\
            (n_neighbors=k, \
             weights=weight_function)
    # print the hyperparameterizations
    print(knn.get_params())
```

Simple Demonstration of the Grid Search Strategy

- The output will be as follows:

Refer to the file 8_3.txt

GridSearchCV

- GridsearchCV is a method of tuning wherein the model can be built by evaluating the combination of parameters mentioned in a grid.
- In the following figure, we will see how GridSearchCV is different from manual search and look at grid search in a muchdetailed way in a table format.

Tuning using GridSearchCV

- For the sake of comparison, we will use the same breast cancer dataset and k-NN classifier as before:

```
from sklearn import model_selection, datasets,  
neighbors  
# load the data  
cancer = datasets.load_breast_cancer()  
# target  
y = cancer.target  
# features  
X = cancer.data
```

Tuning using GridSearchCV

- The next thing we need to do after loading the data is to initialize the class of the estimator we would like to evaluate under different hyperparameterizations:

```
# initialize the estimator  
knn = neighbors.KNeighborsClassifier()
```

Tuning using GridSearchCV

- We then define the grid:

```
# grid contains k and the weight function  
grid = {'n_neighbors': [1, 3, 5, 7],  
        'weights': ['uniform', 'distance']}
```

Tuning using GridSearchCV

- The last thing to do is set the number splits to be used using cross-validation via the cv argument.
- We will set this to 10, thereby conducting 10-fold cross-validation:

!!!!

set up the grid search with scoring on precision and
number of folds = 10

!!!!

```
gscv = model_selection.GridSearchCV(estimator=knn, \
                                      param_grid=grid, \
                                      scoring='precision', cv=10)
```

Tuning using GridSearchCV

- The last step is to feed data to this object via its fit() method.
- Once this has been done, the grid search process will be kick-started:

```
# start the search  
gscv.fit(X, y)
```

Tuning using GridSearchCV

```
GridSearchCV(cv=10, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                             metric='minkowski',
                                             metric_params=None, n_jobs=None,
                                             n_neighbors=5, p=2,
                                             weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'n_neighbors': [1, 3, 5, 7],
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='precision', verbose=0)
```

Tuning using GridSearchCV

- Although not usually a consideration for smaller datasets, this value can be important because in some cases
- You may find that a marginal increase in model performance through a certain hyperparameterization is associated with a significant increase in model fit time, which, depending on the computing resources you have available, may render that hyperparameterization infeasible because it will take too long to fit:

```
# view the results  
print(gscv.cv_results_)
```

Tuning using GridSearchCV

```
{'mean_fit_time': array([0.00055282, 0.00043252, 0.00045586, 0.00044289, 0.00042443,
   0.00044796, 0.00044997, 0.0004324 ]), 'std_fit_time': array([2.26088859e-04, 3.34712853e-06, 4.32680787e-05,
6.06286024e-05,
   1.17754644e-05, 4.16436731e-05, 3.92637181e-05, 1.40615227e-05]), 'mean_score_time': array([0.00221133, 0.0010
2084, 0.00203059, 0.00103009, 0.00206087,
   0.00111656, 0.0021332 , 0.00115821]), 'std_score_time': array([3.21795421e-04, 2.84032587e-05, 7.50346451e-05,
3.56917072e-05,
   6.77137685e-05, 9.23788651e-05, 6.50428475e-05, 9.99759607e-05]), 'param_n_neighbors': masked_array(data=[1,
1, 3, 3, 5, 5, 7, 7],
   mask=[False, False, False, False, False, False, False, False],
   fill_value='?'),
   dtype=object), 'param_weights': masked_array(data=['uniform', 'distance', 'uniform', 'distance',
   'uniform', 'distance', 'uniform', 'distance'],
   mask=[False, False, False, False, False, False, False, False],
   fill_value='?'),
   dtype=object), 'params': [{n_neighbors: 1, weights: 'uniform'}, {n_neighbors: 1, weights: 'distan
ce'}, {n_neighbors: 3, weights: 'uniform'}, {n_neighbors: 3, weights: 'distance'}, {n_neighbors: 5, weight
s: 'uniform'}, {n_neighbors: 5, weights: 'distance'}, {n_neighbors: 7, weights: 'uniform'}, {n_neighbors:
7, weights: 'distance'}], 'split0_test_score': array([0.92105263, 0.92105263, 0.8974359 , 0.8974359 ,
0.91891892,
```

Tuning using GridSearchCV

```
ce'}, {'n_neighbors': 3, 'weights': 'uniform'}, {'n_neighbors': 3, 'weights': 'distance'}, {'n_neighbors': 5, 'weight  
s': 'uniform'}, {'n_neighbors': 5, 'weights': 'distance'}, {'n_neighbors': 7, 'weights': 'uniform'}, {'n_neighbors':  
7, 'weights': 'distance'}], 'split0_test_score': array([0.92105263, 0.92105263, 0.8974359 , 0.8974359 , 0.91891892,  
0.91891892, 0.92105263, 0.92105263]), 'split1_test_score': array([0.86486486, 0.86486486, 0.83333333, 0.833333  
33, 0.85365854,  
0.85365854, 0.83333333, 0.83333333]), 'split2_test_score': array([0.92105263, 0.92105263, 0.91666667, 0.918918  
92, 0.91666667,  
0.91891892, 0.91891892, 0.91891892]), 'split3_test_score': array([0.92105263, 0.92105263, 0.94594595, 0.945945  
95, 0.94736842,  
0.94736842, 0.94736842, 0.94736842]), 'split4_test_score': array([0.8974359 , 0.8974359 , 0.94594595, 0.945945  
95, 0.94594595,  
0.94594595, 0.92105263, 0.92105263]), 'split5_test_score': array([0.91666667, 0.91666667, 0.94594595, 0.945945  
95, 0.94444444,  
0.94444444, 0.94444444, 0.94444444]), 'split6_test_score': array([0.92105263, 0.92105263, 0.94736842, 0.947368  
42, 0.97222222,  
0.94594595, 0.97222222, 0.94594595]), 'split7_test_score': array([0.94444444, 0.94444444, 0.94444444, 0.944444  
44, 0.91891892,  
0.91891892, 0.94285714, 0.94285714]), 'split8_test_score': array([0.93939394, 0.93939394, 0.96875 , 0.969696  
97, 0.96875 ,  
0.94117647, 0.96875 , 0.94117647]), 'split9_test_score': array([0.94444444, 0.94444444, 0.94285714, 0.942857  
14, 0.97142857,  
0.97142857, 0.94444444, 0.94444444]), 'mean_test_score': array([0.91892952, 0.91892952, 0.92852418, 0.928843  
, 0.935556744,  
0.93044707, 0.93114526, 0.92579928]), 'std_test_score': array([0.02271376, 0.02271376, 0.03697908, 0.03701478,  
0.03431266,  
0.0302332 , 0.03733588, 0.0330005 ]), 'rank_test_score': array([7, 7, 5, 4, 1, 3, 2, 6], dtype=int32)}
```

Tuning using GridSearchCV

```
import pandas as pd
# convert the results dictionary to a dataframe
results = pd.DataFrame(gscv.cv_results_)
"""
select just the hyperparameterizations tried,
the mean test scores, order by score and show the top 5 models
"""
print(results.loc[:,['params','mean_test_score']]\
      .sort_values('mean_test_score', ascending=False).head(5))
```

Tuning using GridSearchCV

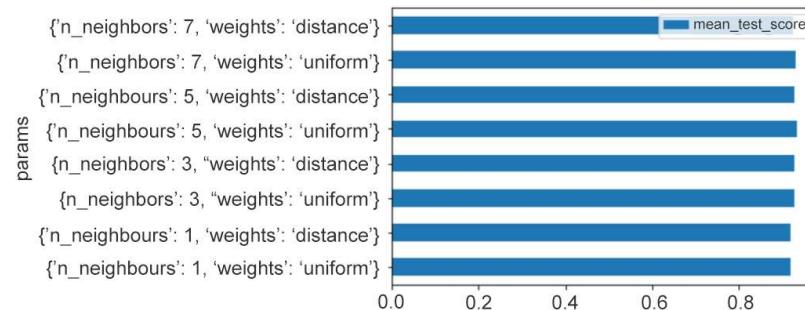
	params	mean_test_score
4	{'n_neighbors': 5, 'weights': 'uniform'}	0.935567
6	{'n_neighbors': 7, 'weights': 'uniform'}	0.931145
5	{'n_neighbors': 5, 'weights': 'distance'}	0.930447
3	{'n_neighbors': 3, 'weights': 'distance'}	0.928843
2	{'n_neighbors': 3, 'weights': 'uniform'}	0.928524

Tuning using GridSearchCV

- We can also use pandas to produce visualizations of the result as follows:

```
# visualise the result  
results.loc[:,['params','mean_test_score']]|\n    .plot.barh(x = 'params')
```

- The output will be as follows:



Support Vector Machine (SVM) Classifiers

- The SVM classifier is basically a supervised machine learning model.
- It is a commonly used class of estimator that can be used for both binary and multi-class classification.
- It is known to perform well in cases where the data is limited, hence it is a reliable model.

Complete Exercise 8.02: Grid Search Hyperparameter Tuning for an SVM

Advantages and Disadvantages of Grid Search

- The primary advantage of the grid search compared to a manual search is that it is an automated process that one can simply set and forget.
- Additionally, you have the power to dictate the exact hyperparameterizations evaluated, which can be a good thing when you have prior knowledge of what kind of hyperparameterizations might work well in your context.

Random Search

- Instead of searching through every hyperparameterizations in a pre-defined set, as is the case with a grid search, in a random search we sample from a distribution of possibilities by assuming each hyperparameter to be a random variable.
- Before we go through the process in depth, it will be helpful to briefly review what random variables are and what we mean by a distribution.

Random Variables and Their Distributions

- A random variable is non-constant (its value can change) and its variability can be described in terms of distribution.
- There are many different types of distributions, but each falls into one of two broad categories: discrete and continuous.
- We use discrete distributions to describe random variables whose values can take only whole numbers, such as counts.

Random Variables and Their Distributions

- For example, if there are 10 possible values the probability that X is any particular value is exactly $1/10$.
- If there were 6 possible values, as in the case of a standard 6-sided die, the probability would be $1/6$, and so on.
- The probability mass function for the discrete uniform distribution is:

$$P(X = x) = \frac{1}{n}$$

Random Variables and Their Distributions

- First, we create a list of all the possible values X can take:

```
# list of all xs  
X = list(range(1, 11))  
print(X)
```

- The output will be as follows:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Random Variables and Their Distributions

- We then calculate the probability that X will take up any value of x ($P(X=x)$):

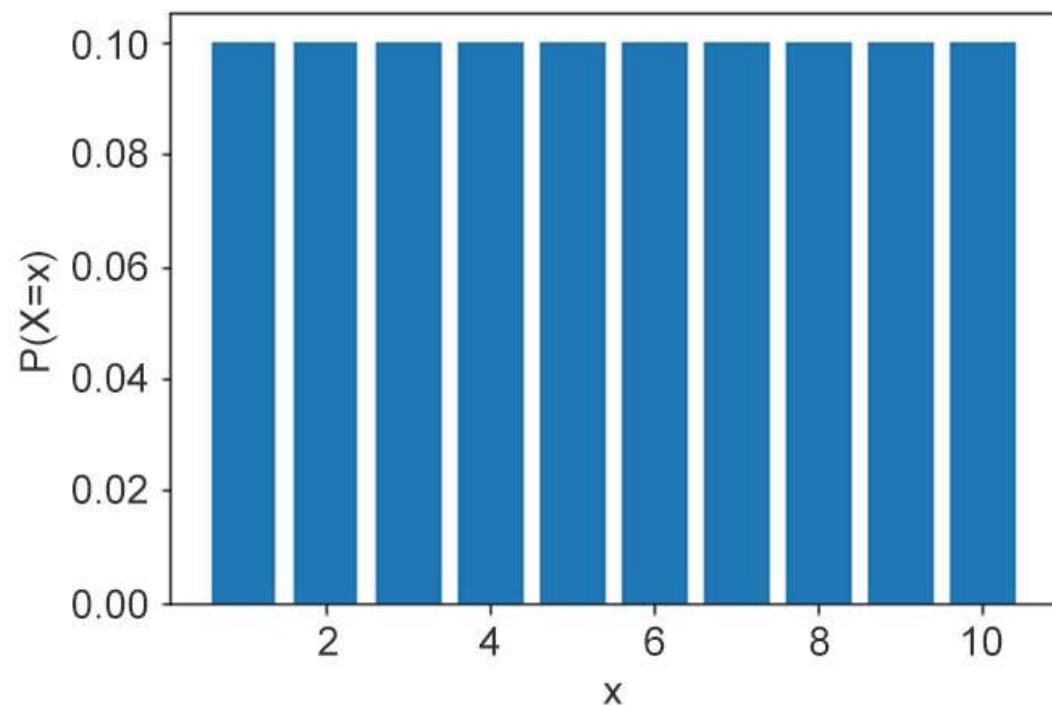
```
# pmf, 1/n * n = 1  
p_X_x = [1/len(X)] * len(X)  
# sums to 1  
print(p_X_x)
```

Random Variables and Their Distributions

- As discussed, the summation of probabilities will equal 1, and this is the case with any distribution.
- We now have everything we need to visualize the distribution:

```
import matplotlib.pyplot as plt  
plt.bar(X, p_X_x)  
plt.xlabel('X')  
plt.ylabel('P(X=x)')
```

Random Variables and Their Distributions



Random Variables and Their Distributions

- The normal distribution has two parameters that describe its shape, mean (μ) and variance (σ^2).
- The probability density function for the normal distribution is:

$$P(X) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Random Variables and Their Distributions

- Let's first generate 100 evenly spaced values from -10 to 10 using NumPy's .linspace method:

```
import numpy as np  
# range of xs  
x = np.linspace(-10, 10, 100)
```

Random Variables and Their Distributions

- Using `scipy.stats` is a good way to work with distributions, and its `pdf` method allows us to easily visualize the shape of probability density functions:

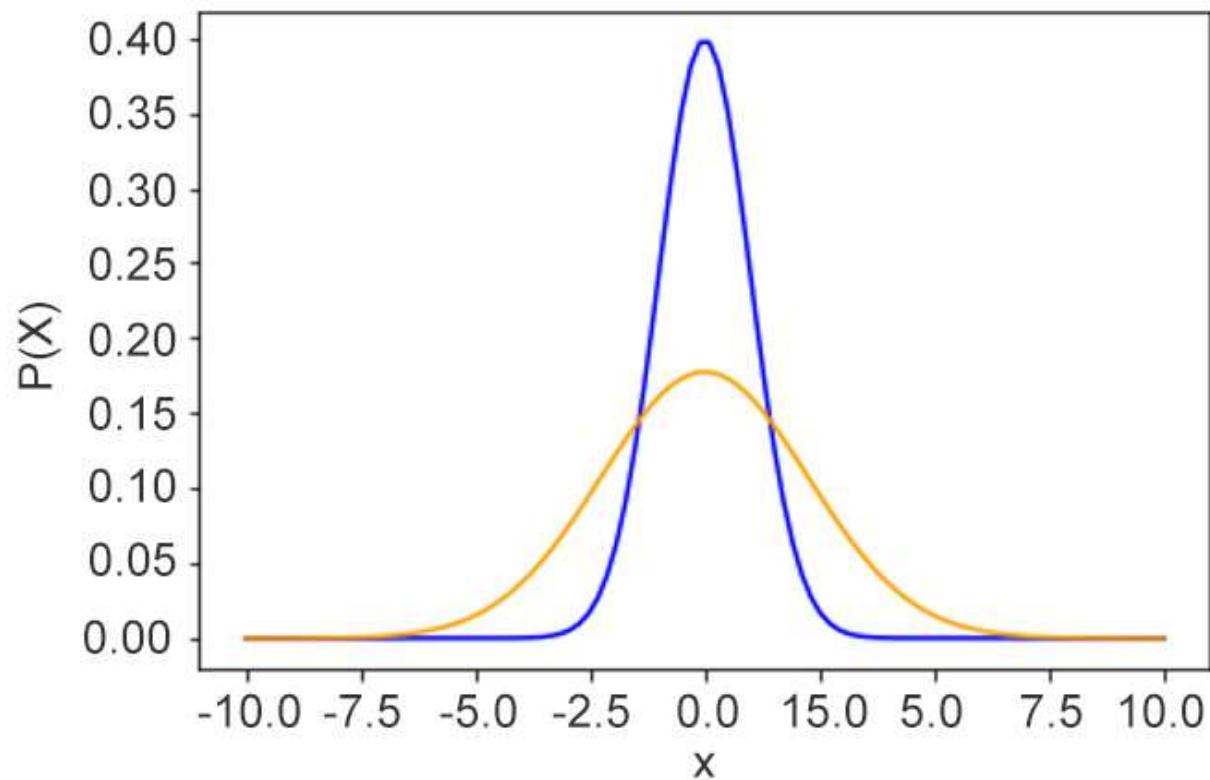
```
import scipy.stats as stats  
# first normal distribution with mean = 0, variance = 1  
p_X_1 = stats.norm.pdf(x=x, loc=0.0, scale=1.0**2)  
# second normal distribution with mean = 0, variance = 2.25  
p_X_2 = stats.norm.pdf(x=x, loc=0.0, scale=1.5**2)
```

Random Variables and Their Distributions

- We then visualize the result.
- Notice that σ^2 controls how fat the distribution is and therefore how variable the random variable is:

```
plt.plot(x,p_X_1, color='blue')
plt.plot(x, p_X_2, color='orange')
plt.xlabel('X')
plt.ylabel('P(X)')
```

Random Variables and Their Distributions



Simple Demonstration of the Random Search Process

- Again, before we get to the scikit-learn implementation of random search parameter tuning, we will step through the process using simple Python tools.
- Up until this point, we have only been using classification problems to demonstrate tuning concepts, but now we will look at a regression problem.

Simple Demonstration of the Random Search Process

- We first load the data:

```
from sklearn import datasets, linear_model,  
model_selection  
# load the data  
diabetes = datasets.load_diabetes()  
# target  
y = diabetes.target  
# features  
X = diabetes.data
```

Simple Demonstration of the Random Search Process

- To get a feel for the data, we can examine the disease progression for the first patient:

```
# the first patient has index 0  
print(y[0])
```

- The output will be as follows:

151.0

Simple Demonstration of the Random Search Process

- Let's now examine their characteristics:

```
# let's look at the first patients data
```

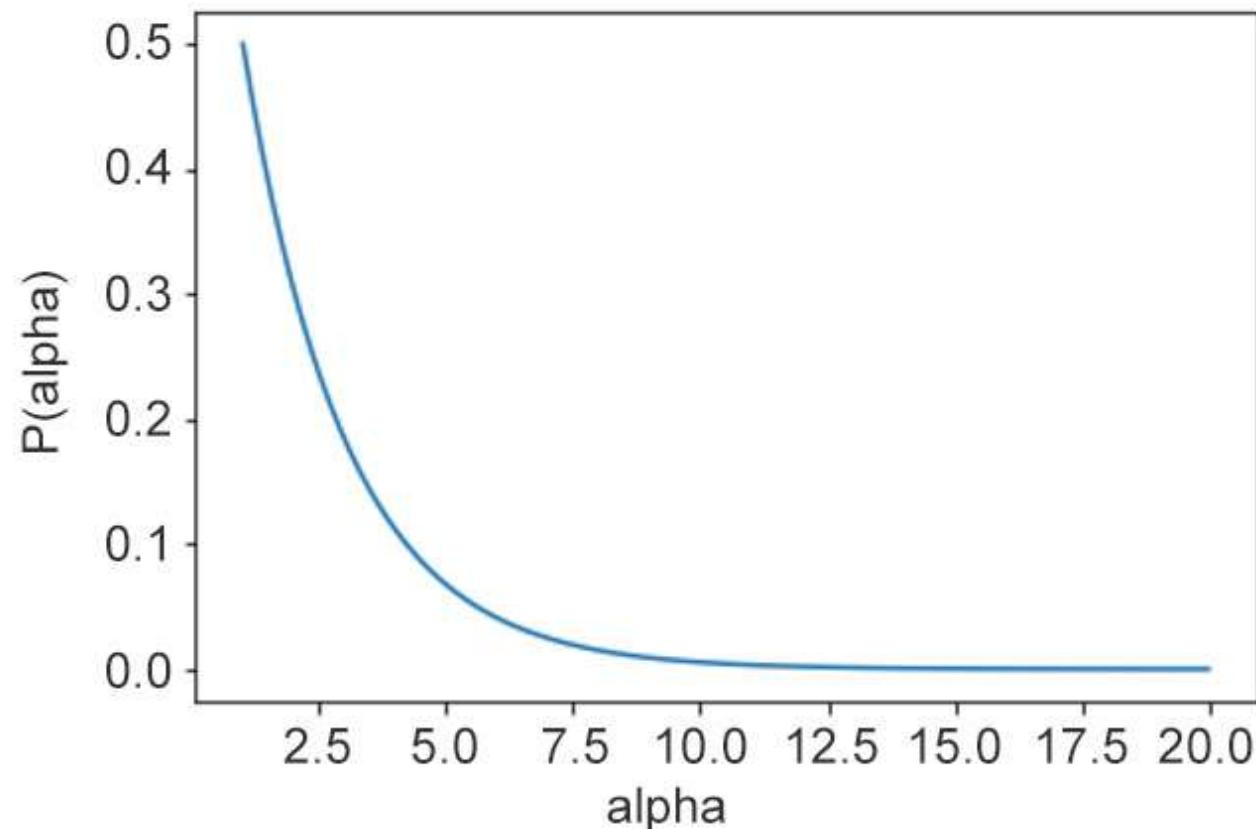
```
print(dict(zip(diabetes.feature_names, X[0])))
```

- We should see the following:

```
{'age': 0.0380759064334241, 'sex': 0.0506801187398187, 'bmi': 0.0616962065186885, 'bp': 0.021872354994955  
8, 's1': -0.0442234984244464, 's2': -0.0348207628376986, 's3': -0.0434008456520269, 's4': -0.0025922619981  
8282, 's5': 0.0199084208763183, 's6': -0.0176461251598052}
```

Simple Demonstration of the Random Search Process

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
# values of alpha
x = np.linspace(1, 20, 100)
# probabilities
p_X = stats.gamma.pdf(x=x, a=1, loc=1, scale=2)
plt.plot(x,p_X)
plt.xlabel('alpha')
plt.ylabel('P(alpha)')
```



Simple Demonstration of the Random Search Process

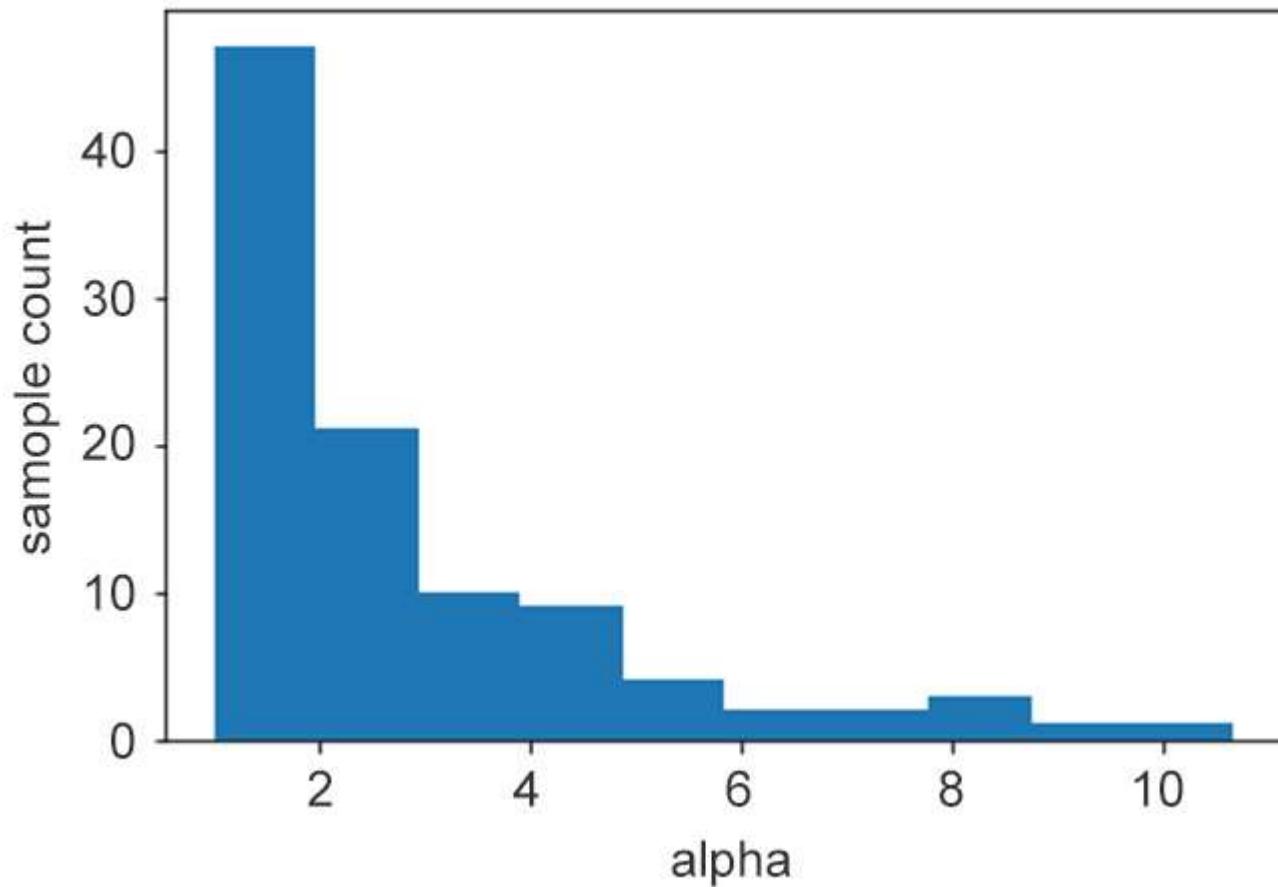
- Remember that the probability of drawing out a particular value of α is related to its probability as defined by this distribution:

```
# n sample values  
n_iter = 100  
# sample from the gamma distribution  
samples = stats.gamma.rvs(a=1, loc=1, scale=2, \  
                         size=n_iter, random_state=100)
```

Simple Demonstration of the Random Search Process

- Note that as your sample sizes increases, the more the histogram conforms to the distribution:

```
# visualize the sample distribution  
plt.hist(samples)  
plt.xlabel('alpha')  
plt.ylabel('sample count')
```



Simple Demonstration of the Random Search Process

- Using this metric means larger values are better.
- We will store the results in a dictionary with each α value as the key and the corresponding cross-validated negative MSE as the value:

Refer to the file 8_4.txt

Simple Demonstration of the Random Search Process

```
import pandas as pd
"""
convert the result dictionary to a pandas dataframe,
transpose and reset the index
"""
df_result = pd.DataFrame(result).T.reset_index()
# give the columns sensible names
df_result.columns = ['alpha', 'mean_neg_mean_squared_error']
print(df_result.sort_values('mean_neg_mean_squared_error', \
                           ascending=False).head())
```

Simple Demonstration of the Random Search Process

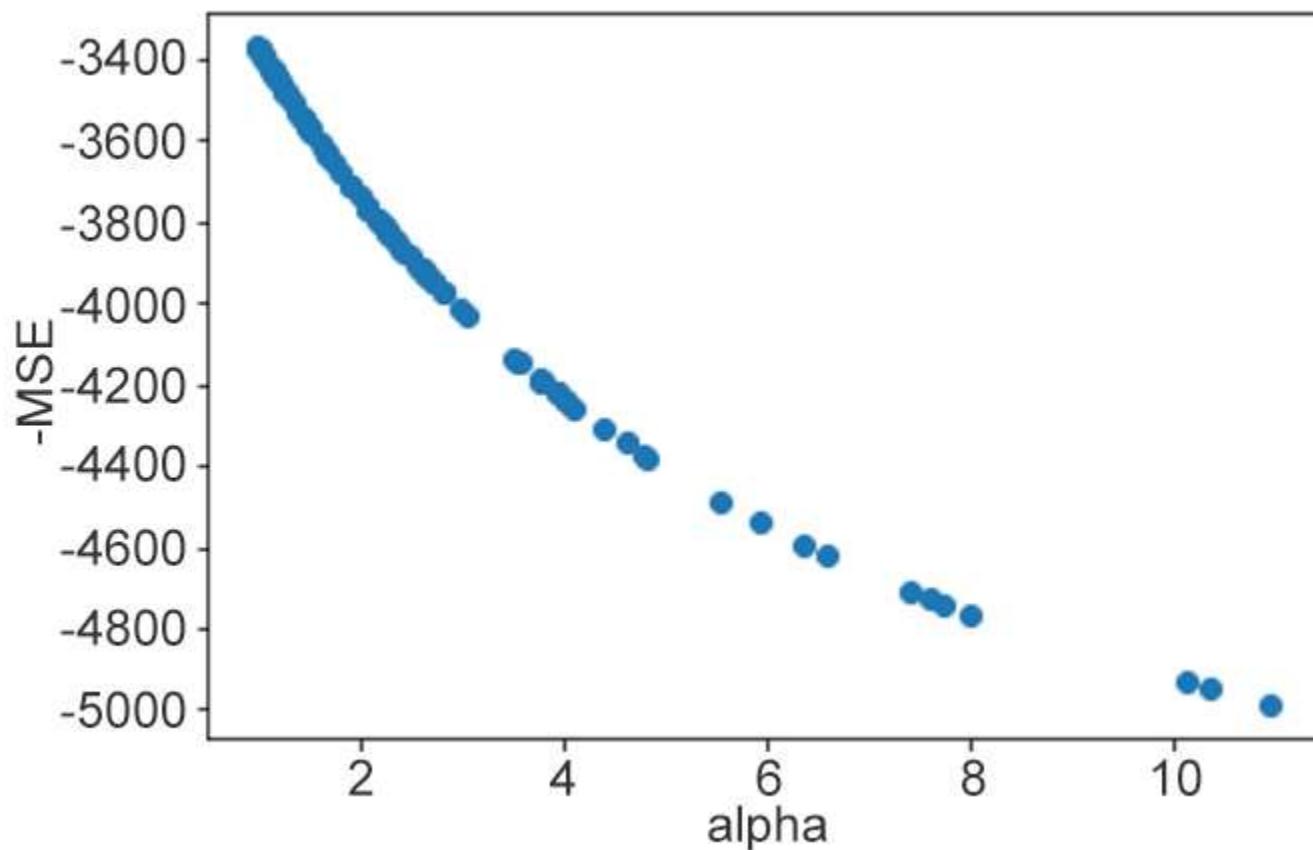
- The output will be as follows:

	alpha	mean_neg_mean_squared_error
4	1.009460	-3368.572167
26	1.011409	-3369.403727
29	1.030745	-3377.629026
43	1.041302	-3382.102466
34	1.074316	-3396.012056

Simple Demonstration of the Random Search Process

- It is always beneficial to visualize results where possible.
- Plotting α by negative mean squared error as a scatter plot makes it clear that venturing away from $\alpha = 1$ does not result in improvements in predictive performance:

```
plt.scatter(df_result.alpha, \
            df_result.mean_neg_mean_squared_error)
plt.xlabel('alpha')
plt.ylabel('-MSE')
```



Tuning Using RandomizedSearchCV

```
from sklearn import datasets, model_selection, linear_model
# load the data
diabetes = datasets.load_diabetes()
# target
y = diabetes.target
# features
X = diabetes.data
# initialise the ridge regression
reg = linear_model.Ridge()
```

Tuning Using RandomizedSearchCV

- We then specify that the hyperparameter we would like to tune is alpha and that we would like α to be distributed gamma, with $k = 1$ and $\theta = 1$:

```
from scipy import stats  
# alpha ~ gamma(1,1)  
param_dist = {'alpha': stats.gamma(a=1, loc=1, scale=2)}
```

Tuning Using RandomizedSearchCV

- Next, we set up and run the random search process, which will sample 100 values from our $\text{gamma}(1,1)$ distribution
- Fit the ridge regression, and evaluate its performance using cross-validation scored on the negative mean squared error metric:

Refer to the file 8_5.txt

Tuning Using RandomizedSearchCV

- Sorting by `rank_test_score` and viewing the first five rows aligns with our conclusion that alpha should be set to 1 and regularization does not seem to be required for this problem:

```
import pandas as pd
# convert the results dictionary to a pandas data frame
results = pd.DataFrame(rscv.cv_results_)
# show the top 5 hyperparamaterizations
print(results.loc[:,['params','rank_test_score']]\n      .sort_values('rank_test_score').head(5))
```

Tuning Using RandomizedSearchCV

- The output will be as follows:

	params	rank_test_score
47	{'alpha': 1.082807253775902}	1
72	{'alpha': 1.0996845885124735}	2
18	{'alpha': 1.1006233499723193}	3
52	{'alpha': 1.1012596885127424}	4
59	{'alpha': 1.1267717423087455}	5

Complete Exercise 8.03: Random Search Hyperparameter Tuning for a Random Forest Classifier

Advantages and Disadvantages of a Random Search

- Because a random search takes a finite sample from a range of possible hyperparameterizations (`n_iter` in `model_selection.RandomizedSearchCV`)
- It is feasible to expand the range of your hyperparameter search beyond what would be practical with a grid search.
- This is because a grid search has to try everything in the range, and setting a large range of values may be too slow to process.

Complete Activity 8.01: Is the Mushroom Poisonous?

Summary

- In this lesson, we have covered three strategies for hyperparameter tuning based on searching for estimator hyperparameterizations that improve performance.
- The manual search is the most hands-on of the three but gives you a unique feel for the process.
- It is suitable for situations where the estimator in question is simple (a low number of hyperparameters).



9. Interpreting a Machine Learning Model



Overview

- This lesson will show you how to interpret a machine learning model's results and get deeper insights into the patterns it found.
- By the end of the lesson, you will be able to analyze weights from linear models and variable importance for RandomForest.

Introduction

- Machine learning algorithms are always referred to as black box where we can only see the inputs and outputs and the implementation inside the algorithm is quite opaque, so people don't know what is happening inside.
- With each day that passes, we can sense the elevated need for more transparency in machine learning models.
- In the last few years, we have seen some cases where algorithms have been accused of discriminating against groups of people

Linear Model Coefficients

- In lesson 2, Regression, and lesson 3, Binary Classification, you saw that linear regression models learn function parameters in the form of the following:

$$\hat{y} = f(X) = w_0 + w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n$$

Linear Model Coefficients

- Let's implement this on a real example with the Diabetes dataset from sklearn:

```
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
data = load_diabetes()
# fit a linear regression model to the data
lr_model = LinearRegression()
lr_model.fit(data.data, data.target)
lr_model.coef_
```

Linear Model Coefficients

- The output will be as follows:

```
array([-10.01219782, -239.81908937,  519.83978679,  324.39042769,
       -792.18416163,   476.74583782,  101.04457032,  177.06417623,
      751.27932109,   67.62538639])
```

Linear Model Coefficients

- Let's create a DataFrame with these values and column names:

```
import pandas as pd  
coeff_df = pd.DataFrame()  
coeff_df['feature'] = data.feature_names  
coeff_df['coefficient'] = lr_model.coef_  
coeff_df.head()
```

Linear Model Coefficients

	feature	coefficient
0	age	-10.012198
1	sex	-239.819089
2	bmi	519.839787
3	bp	324.390428
4	s1	-792.184162

Complete Exercise 9.01: Extracting the Linear Regression Coefficient

RandomForest Variable Importance

- Let's see how to extract this information from the Breast Cancer dataset from sklearn:

```
from sklearn.datasets import load_breast_cancer  
from sklearn.ensemble import RandomForestClassifier  
data = load_breast_cancer()  
X, y = data.data, data.target  
rf_model = RandomForestClassifier(random_state=168)  
rf_model.fit(X, y)  
rf_model.feature_importances_
```

RandomForest Variable Importance

- The output will be as shown here:

```
array([0.03854056, 0.02053266, 0.04877917, 0.06475535, 0.00611284,  
      0.00627651, 0.06487147, 0.10810024, 0.0037876 , 0.00390689,  
      0.01179614, 0.00535536, 0.0040302 , 0.03232201, 0.00470869,  
      0.00466341, 0.00708819, 0.00341199, 0.00282266, 0.00505392,  
      0.11991176, 0.01938264, 0.10187395, 0.10452031, 0.01187623,  
      0.01984074, 0.03877298, 0.12437077, 0.0055569 , 0.00697787])
```

RandomForest Variable Importance

- Let's create a DataFrame that will contain these values with the name of the columns:

```
import pandas as pd  
varimp_df = pd.DataFrame()  
varimp_df['feature'] = data.feature_names  
varimp_df['importance'] =  
rf_model.feature_importances_  
varimp_df.head()
```

RandomForest Variable Importance

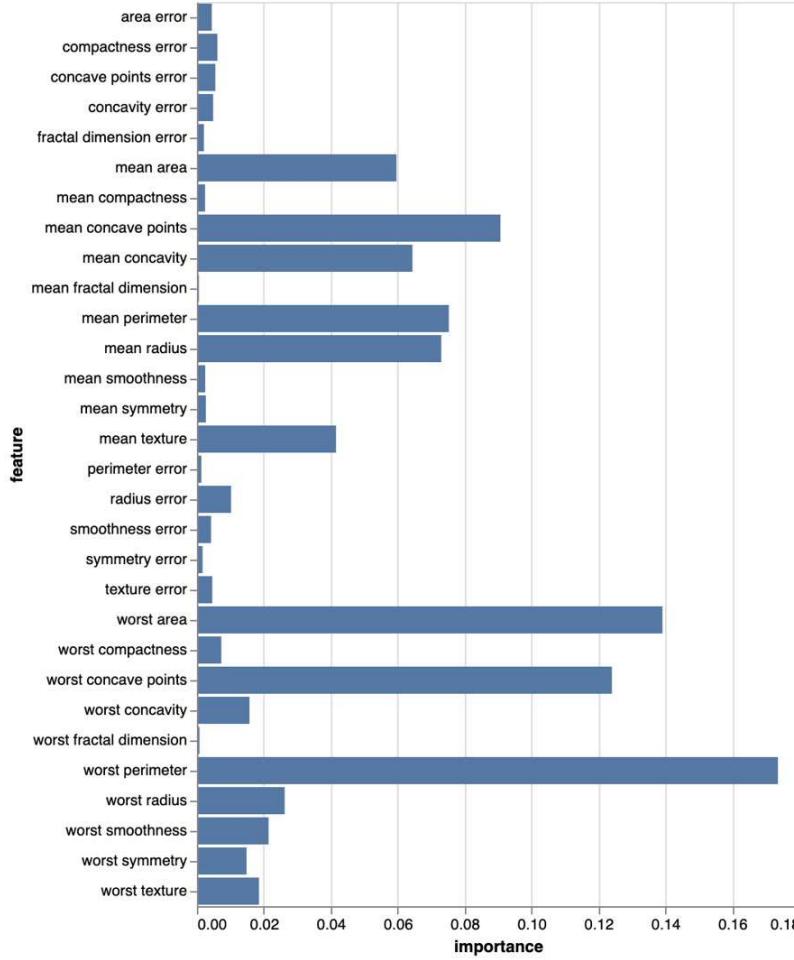
- The output will be as follows:

	feature	importance
0	mean radius	0.073053
1	mean texture	0.041600
2	mean perimeter	0.075311
3	mean area	0.059660
4	mean smoothness	0.002494

RandomForest Variable Importance

- Let's plot these variable importance values onto a graph using altair:

```
import altair as alt  
alt.Chart(varimp_df).mark_bar().encode(x='importance',  
                                         y="feature")
```



Complete Exercise 9.02: Extracting RandomForest Feature Importance

Variable Importance via Permutation

- This technique can be referred to as variable importance via permutation.
- Let's say we trained a model to predict a target variable with five classes and achieved an accuracy of 0.95.
- One way to assess the importance of one of the features is to remove and train a model and see the new accuracy score.
- If the accuracy score dropped significantly, then we could infer that this variable has a significant impact on the prediction.

Variable Importance via Permutation

- For example, we have the following rows in the original dataset:

X1		X2	X3	target
1		0.213	-23	0
42		0.783	-13	0
12		0.0013	-321	1

Variable Importance via Permutation

- We can swap the values for the X1 column and get a new dataset:

X1	X2	X3	Target
42	0.213	-23	0
12	0.783	-13	1
1	0.0013	-321	1

Variable Importance via Permutation

- First, let's load the data and train a Random Forest model:

```
from sklearn.datasets import load_breast_cancer  
from sklearn.ensemble import RandomForestClassifier
```

```
data = load_breast_cancer()  
X, y = data.data, data.target  
rf_model = RandomForestClassifier(random_state=168)  
rf_model.fit(X, y)
```

Variable Importance via Permutation

- Consider the following code snippet:

```
from mlxtend.evaluate import  
feature_importance_permutation  
imp_vals, _ = feature_importance_permutation\  
    (predict_method=rf_model.predict, X=X, y=y, \  
     metric='r2', num_rounds=1, seed=2)  
imp_vals
```

Variable Importance via Permutation

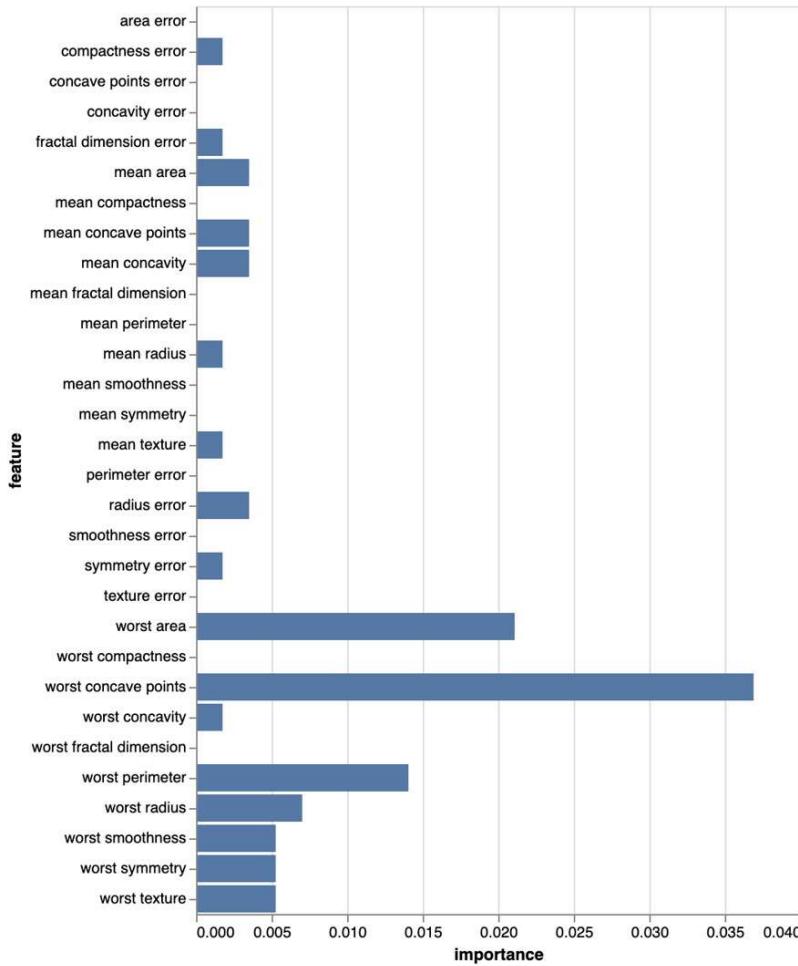
- The output should be as follows:

```
array([0.          , 0.          , 0.          , 0.          , 0.          , 0.          , 0.          ,  
      0.          , 0.          , 0.          , 0.          , 0.0075181, 0.          , 0.          ,  
      0.          , 0.0075181, 0.          , 0.          , 0.          , 0.          , 0.          , 0.          ,  
      0.          , 0.          , 0.          , 0.          , 0.0075181, 0.0075181, 0.          , 0.          ,  
      0.          , 0.          , 0.0075181, 0.          , 0.          , 0.          , 0.          , 0.        ])
```

Linear Model Coefficients

- Let's create a DataFrame containing these values and the names of the features and plot them on a graph with altair:

```
import pandas as pd
varimp_df = pd.DataFrame()
varimp_df['feature'] = data.feature_names
varimp_df['importance'] = imp_vals
varimp_df.head()
import altair as alt
alt.Chart(varimp_df).mark_bar().encode(x='importance',\
                                         y="feature")
```



Complete Exercise 9.03: Extracting Feature Importance via Permutation

Partial Dependence Plots

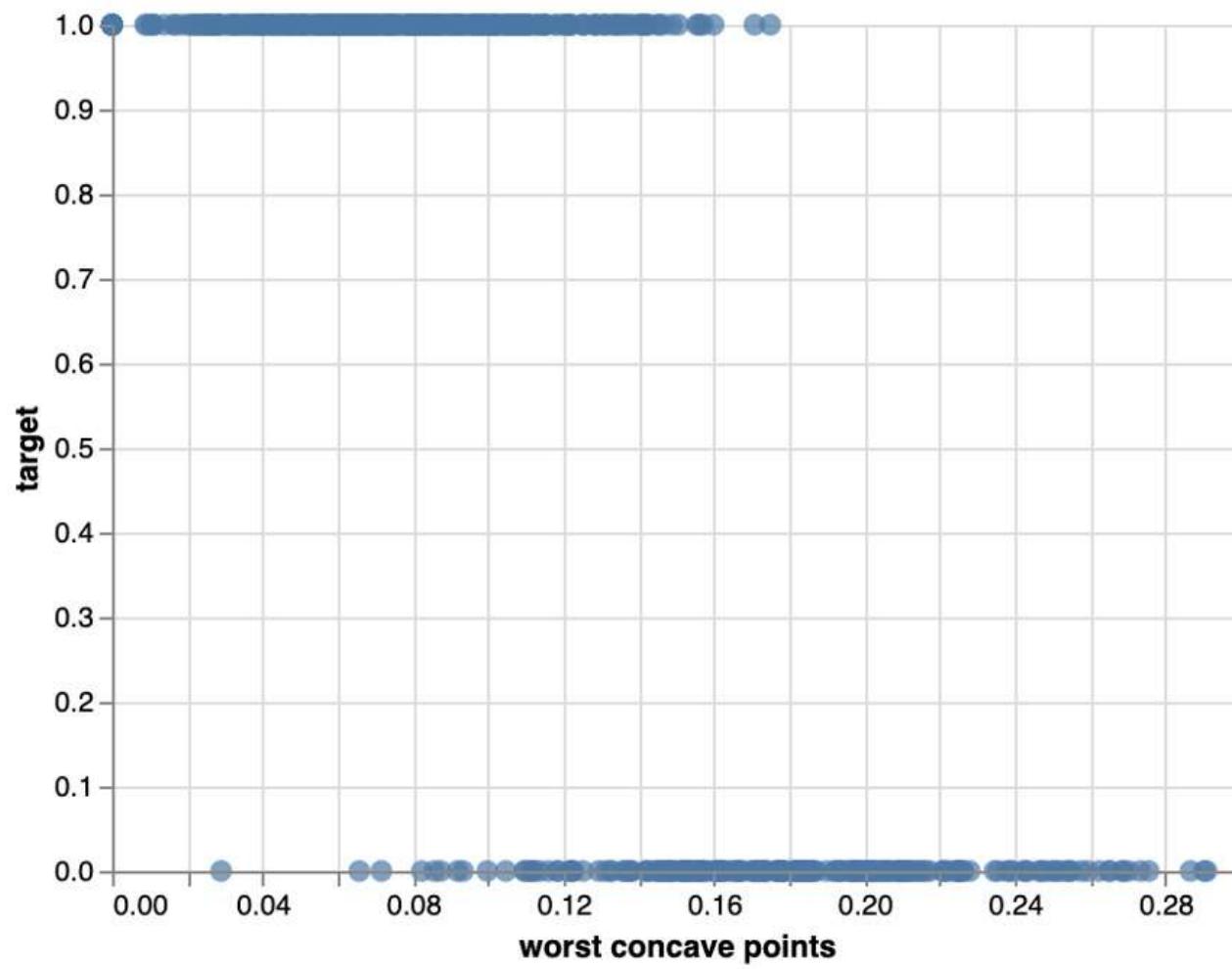
- Let's try it on the Breast Cancer dataset from sklearn:

```
from sklearn.datasets import load_breast_cancer  
import pandas as pd  
data = load_breast_cancer()  
df = pd.DataFrame(data.data,  
columns=data.feature_names)  
df['target'] = data.target
```

Partial Dependence Plots

- Now that we have loaded the data and converted it to a DataFrame, let's have a look at the worst concave points column:

```
import altair as alt  
alt.Chart(df).mark_circle(size=60)\n    .encode(x='worst concave points', y='target')
```



1-449

Original records

Generated records

	X1	X2	X3	target
	1	0.213	-23	0
	42	0.783	-13	0
	12	0.0013	-321	1
	42	0.213	-23	0
	12	0.213	-23	1
	1	0.783	-13	1
	12	0.783	-13	0
	1	0.0013	-321	1
	42	0.0013	-321	1

Partial Dependence Plots

- Let's see how to use it on the Breast Cancer dataset.
- First, we need to get the index of the column we are interested in.
- We will use the `.get_loc()` method from pandas to get the index for the worst concave points column:

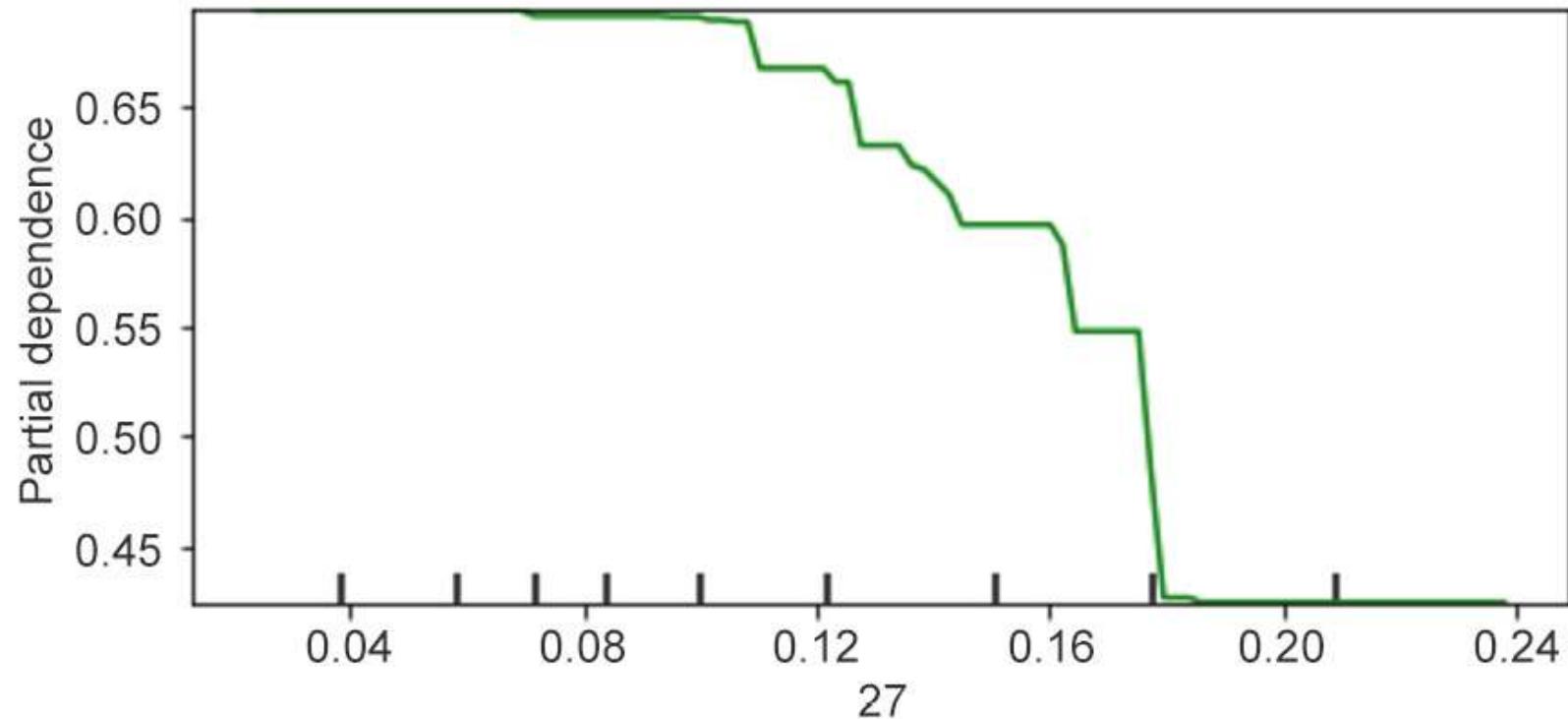
```
import altair as alt
from sklearn.inspection import plot_partial_dependence
feature_index = df.columns.get_loc("worst concave
points")
```

Partial Dependence Plots

- Now we can call the `plot_partial_dependence()` function.
- We need to provide the following parameters: the trained model, the training set, and the indices of the features to be analyzed:

```
plot_partial_dependence(rf_model, df, \
                        features=[feature_index])
```

Partial Dependence Plots



27

1-453

Complete Exercise 9.04: Plotting Partial Dependence

Local Interpretation with LIME

- After training our model, we usually use it for predicting outcomes on unseen data.
- The global interpretations we saw earlier, such as model coefficient, variable importance, and the partial dependence plot, gave us a lot of information on the features at an overall level.
- Sometimes we want to understand what has influenced the model for a specific case to predict a specific outcome.

Local Interpretation with LIME

If we received an observation of $x_1=0$, $x_2=2$ and $x_3=1$, we would know the contribution of:

- x_1 was $0.2 * 0 = 0$
- x_2 was $200 * 2 = +400$
- x_3 was $-180 * 1 = -180$

Local Interpretation with LIME

- First, we will load the data and train a Random Forest model:

```
from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split  
from sklearn.ensemble import RandomForestClassifier  
data = load_breast_cancer()  
X, y = data.data, data.target  
X_train, X_test, y_train, y_test = train_test_split\  
    (X, y, test_size=0.3, \  
     random_state=1)  
rf_model = RandomForestClassifier(random_state=168)  
rf_model.fit(X_train, y_train)
```

Local Interpretation with LIME

- The lime package is not directly accessible on Google Colab, so we need to manually install it with the following command:

```
!pip install lime
```

Local Interpretation with LIME

```
Collecting lime
  Downloading https://files.pythonhosted.org/packages/b5/e0/60070b461a589b2fee0dbc45df9987f150fc83667c2f8a064cef7dbac6b/lime-0.1.1.37.tar.gz (275kB)
    |████████| 276kB 3.5MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from lime) (1.17.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from lime) (1.4.1)
Collecting progressbar
  Downloading https://files.pythonhosted.org/packages/a3/a6/b8e451f6cff1c99b4747a2f7235aa904d2d49e8e1464e0b798272aa84358/progressbar-2.5.tar.gz
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.6/dist-packages (from lime) (0.22.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from lime) (3.1.2)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.6/dist-packages (from lime) (0.16.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn>=0.18->lime) (0.14.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->lime) (2.6.1)
Requirement already satisfied: pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->lime) (2.4.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->lime) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->lime) (0.10.0)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image>=0.12->lime) (2.4)
Requirement already satisfied: pillow>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image>=0.12->lime) (6.2.2)
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image>=0.12->lime) (1.1.1)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.6/dist-packages (from scikit-image>=0.12->lime) (2.4.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1->matplotlib->lime) (1.12.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from kiwisolver>=1.0.1->matplotlib->lime) (42.0.2)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx>=2.0->scikit-image>=0.12->lime) (4.4.1)
Building wheels for collected packages: lime, progressbar
  Building wheel for lime (setup.py) ... done
    Created wheel for lime: filename=lime-0.1.1.37-cp36-none-any.whl size=284277 sha256=4a26b91294e4a192b234e31171003837d9050835dc017b325ae4b0de3bdcbdc
    Stored in directory: /root/.cache/pip/wheels/c1/38/e7/50d75d4fb75afa604570dc42f20c5c5f5ab26d3fbe8d6ef27b
  Building wheel for progressbar (setup.py) ... done
    Created wheel for progressbar: filename=progressbar-2.5-cp36-none-any.whl size=12073 sha256=fba4ccb035d6a61d4a19c2081852a8016d508cf296af67060a32b13
    Stored in directory: /root/.cache/pip/wheels/c0/e9/6b/ea01090205e285175842339aa3b491adeb4015206cd4272ff0
Successfully built lime progressbar
Installing collected packages: progressbar, lime
Successfully installed lime-0.1.1.37 progressbar-2.5
```

Local Interpretation with LIME

- Once installed, we will instantiate the LimeTabularExplainer class by providing the training data, the names of the features, the names of the classes to be predicted, and the task type (in this example, it is classification):

```
from lime.lime_tabular import LimeTabularExplainer  
lime_explainer = LimeTabularExplainer\  
    (X_train, feature_names=data.feature_names,\  
     class_names=data.target_names,\  
     mode='classification')
```

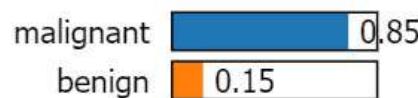
Local Interpretation with LIME

- Finally, we will call the `.show_in_notecourse()` method to display the results from lime:

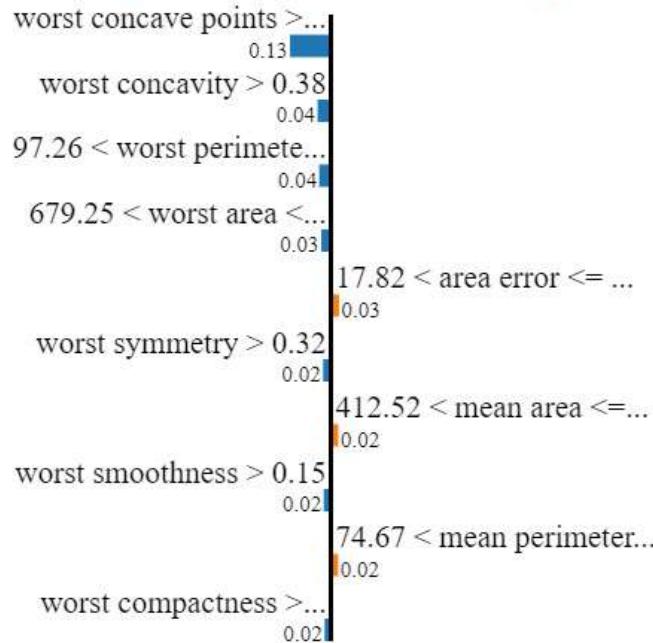
```
exp = lime_explainer.explain_instance\  
      (X_test[1], rf_model.predict_proba,  
       num_features=10)  
exp.show_in_notecourse()
```

Local Interpretation with LIME

Prediction probabilities



malignant



benign

Feature	Value
worst concave points	0.21
worst concavity	0.50
worst perimeter	102.80
worst area	759.40
area error	24.25
worst symmetry	0.39
mean area	534.60
worst smoothness	0.18
mean perimeter	85.98
worst compactness	0.42

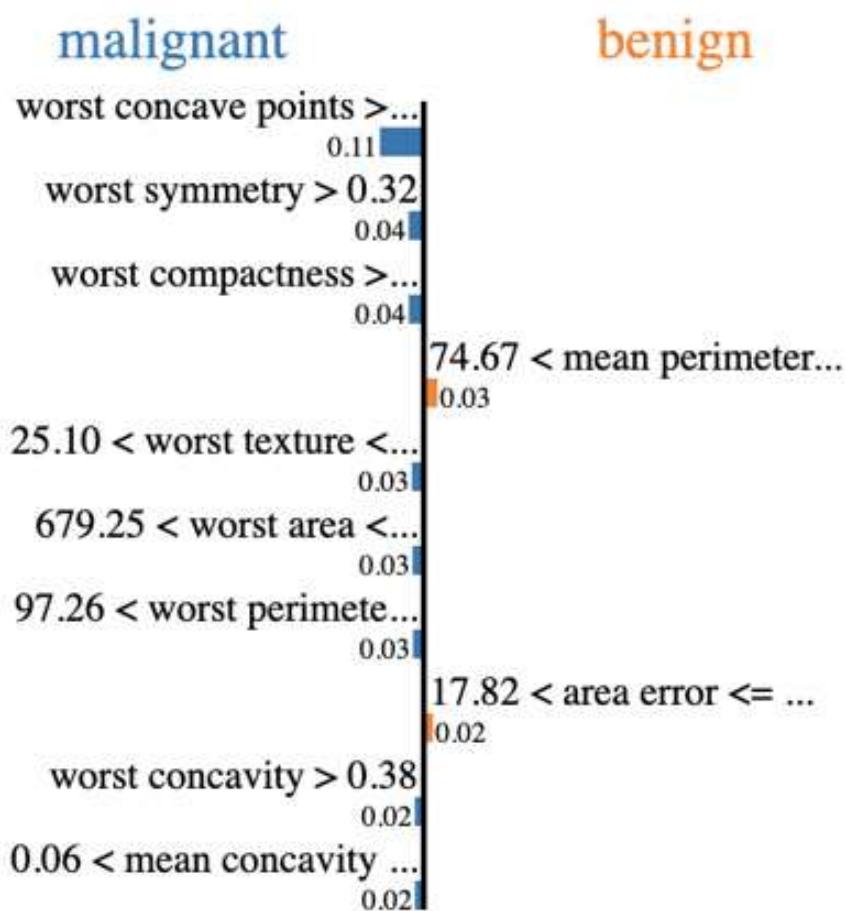
Local Interpretation with LIME

- For this observation, the model thinks there is a 0.85 probability that the predicted value will be malignant:



Local Interpretation with LIME

Feature	Value
worst concave points	0.21
worst symmetry	0.39
worst compactness	0.42
mean perimeter	85.98
worst texture	27.95
worst area	759.40
worst perimeter	102.80
area error	24.25
worst concavity	0.50
mean concavity	0.12



Complete Exercise 9.05: Local Interpretation with LIME

Complete Activity 9.01: Train and Analyze a Network Intrusion Detection Model

Summary

- In this lesson, we learned a few techniques for interpreting machine learning models.
- We saw that there are techniques that are specific to the model used: coefficients for linear models and variable importance for tree-based models.
- There are also some methods that are model-agnostic, such as variable importance via permutation.



10. Analyzing a Dataset



Overview

- By the end of this lesson, you will be able to explain the key steps involved in performing exploratory data analysis
- Identify the types of data contained in the dataset
- Summarize the dataset and at a detailed level for each variable
- Visualize the data distribution in each column; find relationships between variables and analyze missing values and outliers for each variable

Introduction

A very popular methodology that's used in the industry for running data science projects is CRISP-DM and this methodology breaks down a data science project into six different stages:

- Business understanding
- Data understanding
- Data preparation
- Modeling
- Evaluation
- Deployment

Exploring Your Data

- Let's read the data using the `.read_excel()` method and store it in a pandas DataFrame, as shown in the following code snippet:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'data-science/blob/'  
          'master/lesson10/dataset/'  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Exploring Your Data

- After loading the data into a DataFrame, we want to know the size of this dataset, that is, its number of rows and columns.
- To get this information, we just need to call the `.shape` attribute from pandas:
`df.shape`
- You should get the following output:
`(541909, 8)`

Exploring Your Data

- Since this attribute returns a tuple, we can access each of its elements independently by providing the relevant index.
- Let's extract the number of rows (index 0):

`df.shape[0]`

- You should get the following output:

`541909`

Exploring Your Data

- Similarly, we can get the number of columns with the second index:

`df.shape[1]`

- You should get the following output:

8

Exploring Your Data

- Once loaded into a pandas DataFrame, you can print out its content by calling it directly:

`df`

- You should get the following output:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom

Exploring Your Data

- To access the names of the columns for this DataFrame, we can call the .columns attribute:
`df.columns`
- You should get the following output:

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

Exploring Your Data

- Looking at these names, we can potentially guess what types of information are contained in these columns.
- However, to be sure, we can use the `dtypes` attribute, as shown in the following code snippet:

`df.dtypes`

- You should get this output:

```
InvoiceNo          object
StockCode          object
Description        object
Quantity           int64
InvoiceDate        datetime64[ns]
UnitPrice          float64
CustomerID        float64
Country            object
dtype: object
```

Exploring Your Data

- The pandas package provides a single method that can display all the information we have seen so far, that is, the `info()` method:

`df.info()`

- You should get the following output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode       541909 non-null object
Description    540455 non-null object
Quantity        541909 non-null int64
InvoiceDate    541909 non-null datetime64[ns]
UnitPrice       541909 non-null float64
CustomerID     406829 non-null float64
Country         541909 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

Analyzing Your Dataset

- First, we need to import the pandas package:

```
import pandas as pd
```

- Then, we'll load the data into a pandas DataFrame:

```
file_url = 'https://github.com/fenago/\  
    'data-science/blob/\  
    'master/lesson10/dataset/\  
    'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Analyzing Your Dataset

- The `head()` method will show the top rows of your dataset.
- By default, pandas will display the first five rows:
`df.head()`
- You should get the following output:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Analyzing Your Dataset

- Let's try this by displaying the first 10 rows:

`df.head(10)`

- You should get the following output:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047.0	United Kingdom

Analyzing Your Dataset

- Like `head()`, this method, by default, will display only five rows, but you can specify the number of rows you want as a parameter.
- Here, we will display the last eight rows:
`df.tail(8)`
- You should get the following output:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
541901	581587	22367 CHILDRENS APRON SPACEBOY DESIGN	8	2011-12-09 12:50:00	1.95	12680.0	France
541902	581587	22629 SPACEBOY LUNCH BOX	12	2011-12-09 12:50:00	1.95	12680.0	France
541903	581587	23256 CHILDRENS CUTLERY SPACEBOY	4	2011-12-09 12:50:00	4.15	12680.0	France
541904	581587	22613 PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899 CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254 CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255 CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138 BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

Analyzing Your Dataset

- You can also specify a seed (which we covered in lesson 5, Performing Your First Cluster Analysis) in order to get reproducible results if you run the same code again with the `random_state` parameter:
`df.sample(n=5, random_state=1)`
- You should get the following output:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
94801	C544414	22960 JAM MAKING SET WITH JARS	-2	2011-02-18 14:54:00	3.75	13408.0	United Kingdom
210111	555276	48111 DOORMAT 3 SMILEY CATS	1	2011-06-01 17:28:00	15.79	NaN	United Kingdom
455946	575656	22952 60 CAKE CASES VINTAGE CHRISTMAS	48	2011-11-10 14:29:00	0.55	13319.0	United Kingdom
403542	571636	20674 GREEN POLKA DOT BOWL	16	2011-10-18 11:41:00	1.25	13509.0	United Kingdom
471951	576657	22556 PLASTERS IN TIN CIRCUS PARADE	12	2011-11-16 11:03:00	1.65	12720.0	Germany

Complete Exercise 10.01: Exploring the Ames Housing Dataset with Descriptive Statistics

Analyzing the Content of a Categorical Variable

- Now that we've got a good feel for the kind of information contained in the online retail dataset, we want to dig a little deeper into each of its columns:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'data-science/blob'\  
          '/master/lesson10/dataset/'\  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Analyzing the Content of a Categorical Variable

- For instance, we would like to know how many different values are contained in each of the variables by calling the `nunique()` method.
- This is particularly useful for a categorical variable with a limited number of values, such as `Country`:
`df['Country'].nunique()`
- You should get the following output:
38

Analyzing the Content of a Categorical Variable

- Thankfully, the pandas package provides a method to get these results: unique():
`df['Country'].unique()`
- You should get the following output:

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

Analyzing the Content of a Categorical Variable

- By providing the `dropna=False` and `normalize=True` parameters, this method will include the missing value in the listing and calculate the number of occurrences as a ratio, respectively:

```
df['Country'].value_counts(dropna=False,  
normalize=True)
```

United Kingdom	0.914320
Germany	0.017521
France	0.015790
EIRE	0.015124
Spain	0.004674
Netherlands	0.004375
Belgium	0.003818
Switzerland	0.003694
Portugal	0.002803
Australia	0.002323
Norway	0.002004
Italy	0.001482
Channel Islands	0.001399
Finland	0.001283
Cyprus	0.001148
Sweden	0.000853
Unspecified	0.000823
Austria	0.000740
Denmark	0.000718

Complete Exercise 10.02: Analyzing the Categorical Variables from the Ames Housing Dataset

Summarizing Numerical Variables

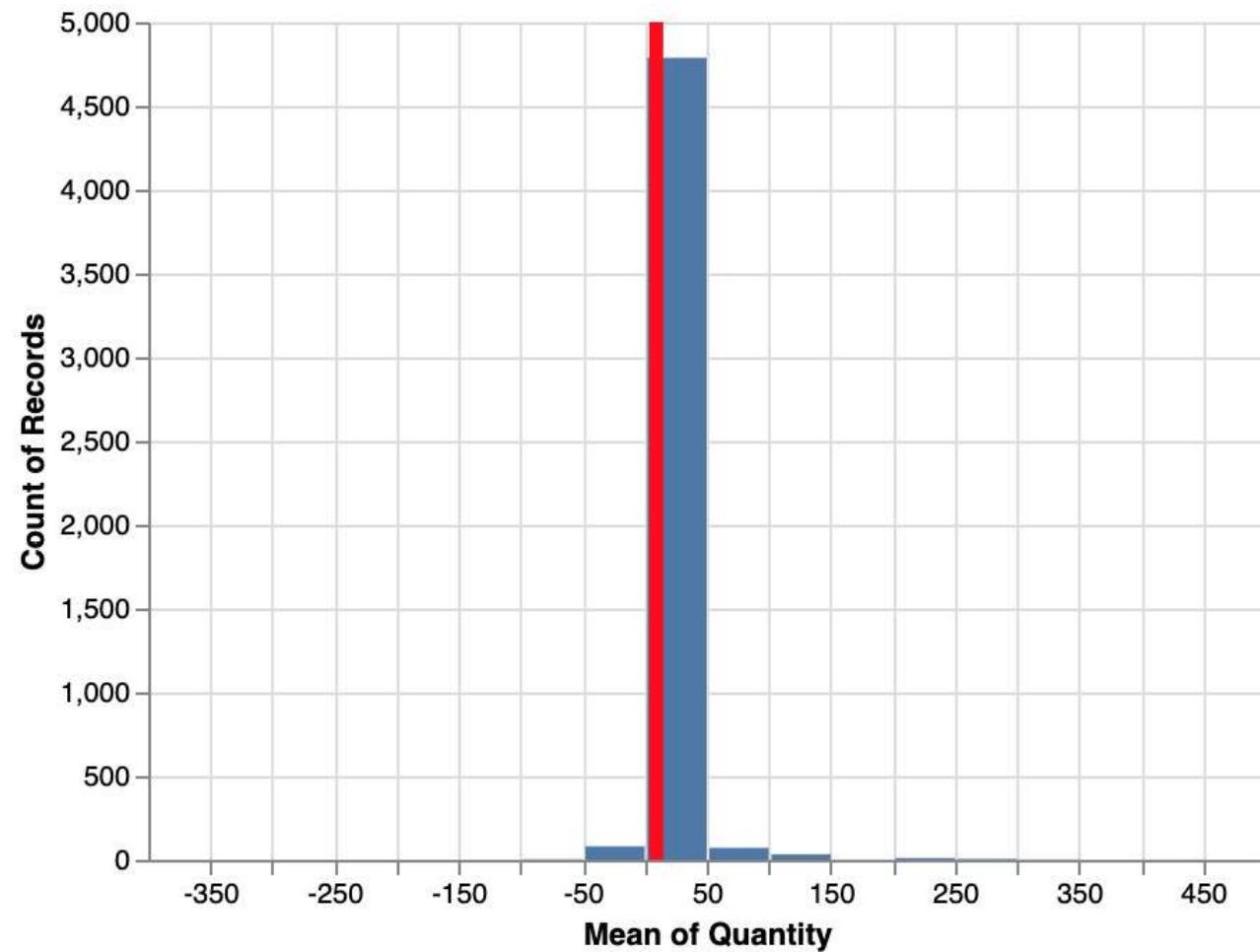
- With the pandas package, a lot of these measures have been implemented as methods.
- For instance, if we want to know what the highest value contained in the 'Quantity' column is, we can use the `.max()` method:

```
df['Quantity'].max()
```

- You should get the following output:
80995

Summarizing Numerical Variables

- Now, let's have a look at the lowest value for 'Quantity' using the `.min()` method:
`df['Quantity'].min()`
- You should get the following output:
`-80995`

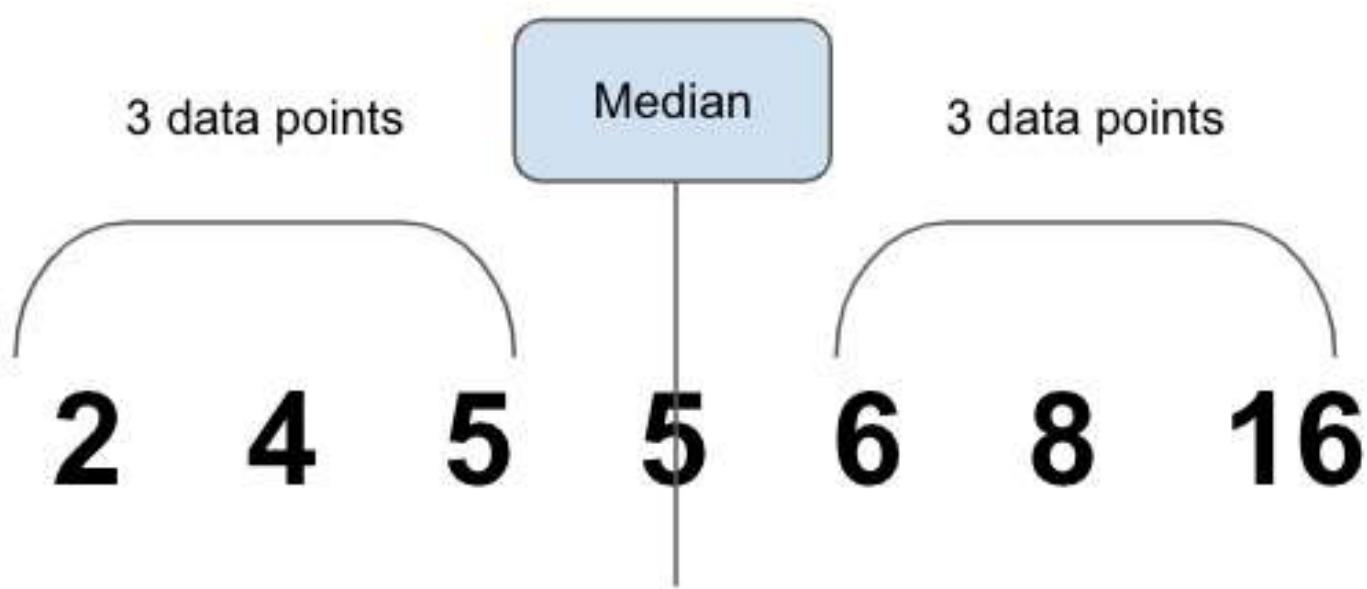


1-494

Summarizing Numerical Variables

- We can get the average value of a feature by using the mean() method from pandas:
`df['Quantity'].mean()`
- You should get the following output:
`9.55224954743324`

Summarizing Numerical Variables



Summarizing Numerical Variables

- In pandas, you can call the median() method to get this value:

```
df['Quantity'].median()
```

- You should get the following output:

3.0

Summarizing Numerical Variables

- We will use the std() method from pandas to calculate this measure:
`df['Quantity'].std()`
- You should get the following output:
`218.08115784986612`

Summarizing Numerical Variables

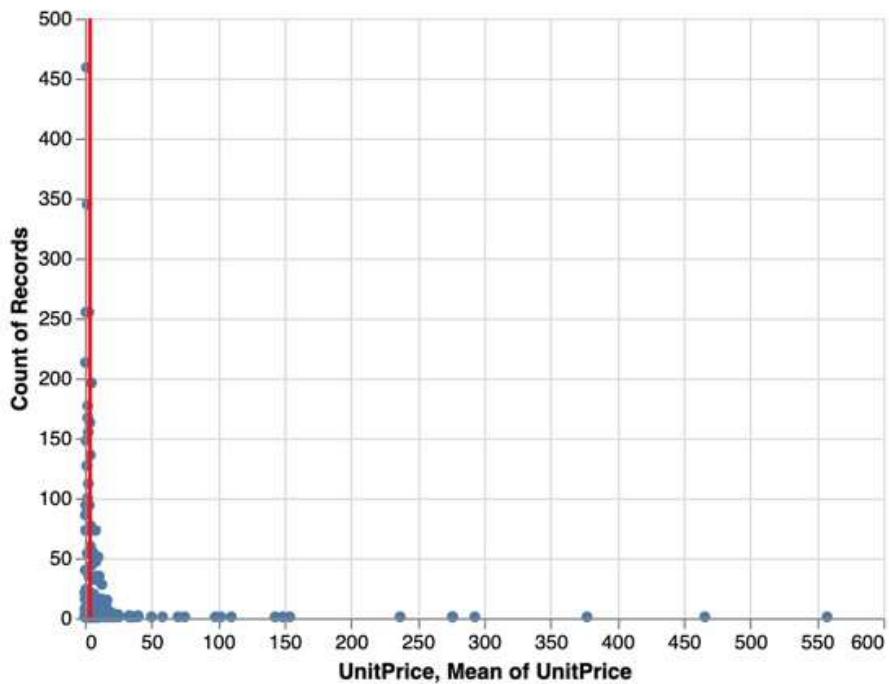
- In the pandas package, there is a method that can display most of these descriptive statistics with one single line of code: `describe()`:
`df.describe()`
- You should get this output:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

Complete Exercise 10.03: Analyzing Numerical Variables from the Ames Housing Dataset

Visualizing Your Data

```
count      5000.000000
mean       3.895058
std        14.928765
min        0.000000
25%       1.250000
50%       2.080000
75%       4.130000
max       557.720000
Name: UnitPrice, dtype: float64
```



Using the Altair API

- Let's see how we can display a bar chart step by step on the online retail dataset.
- First, import the pandas and altair packages:

```
import pandas as pd  
import altair as alt
```

Using the Altair API

- Then, load the data into a pandas DataFrame:

```
file_url = 'https://github.com/fenago/'\
    'data-science/blob/'\
    'master/lesson10/dataset/'\
    'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Using the Altair API

- We will randomly sample 5,000 rows of this DataFrame using the `sample()` method (altair requires additional steps in order to display a larger dataset):
`sample_df = df.sample(n=5000, random_state=8)`
- Now instantiate a Chart object from altair with the pandas DataFrame as its input parameter:
`base = alt.Chart(sample_df)`

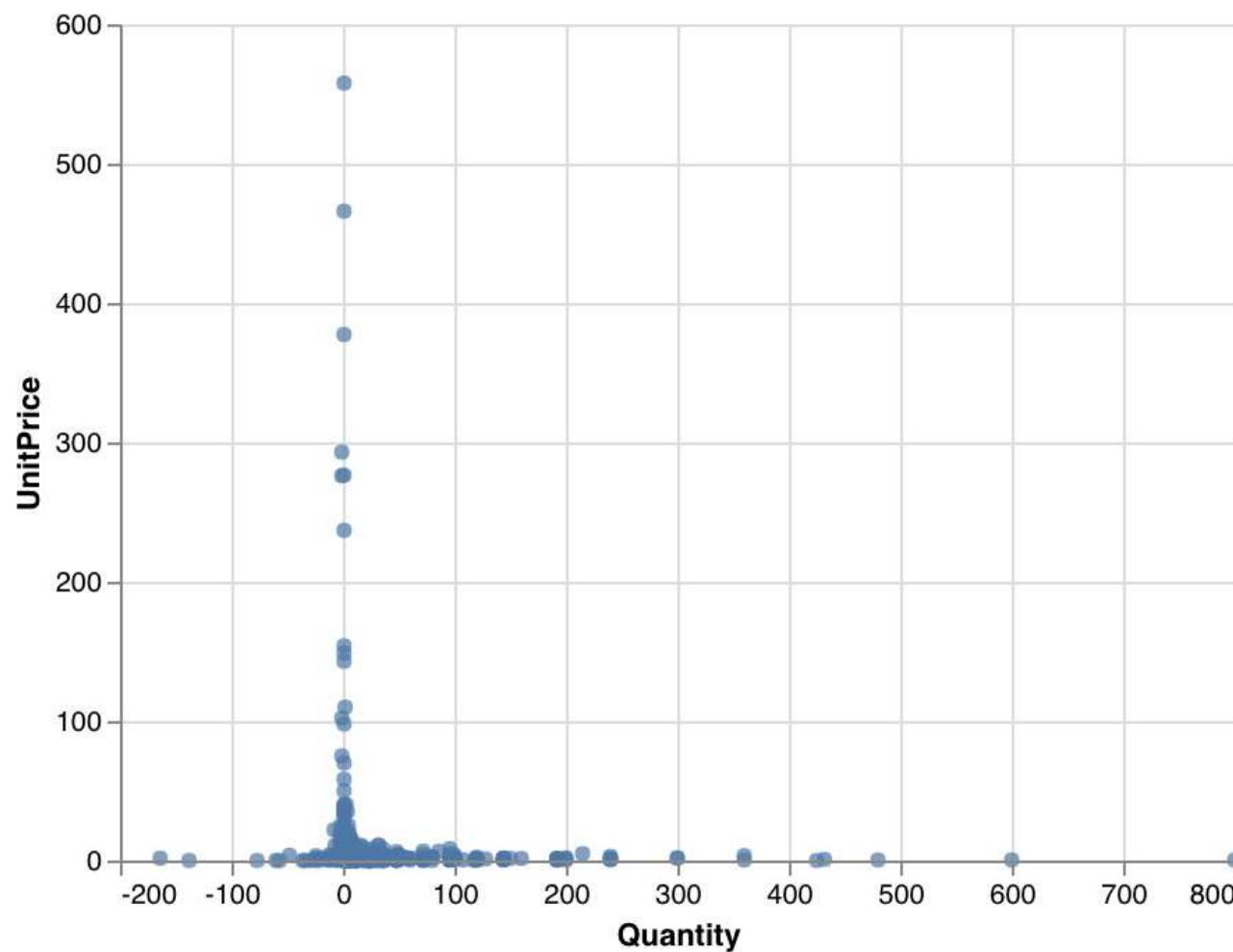
Using the Altair API

- Next, we call the `mark_circle()` method to specify the type of graph we want to plot: a scatter plot:

```
chart = base.mark_circle()
```

- Finally, we specify the names of the columns that will be displayed on the x and y axes using the `encode()` method:

```
chart.encode(x='Quantity', y='UnitPrice')
```

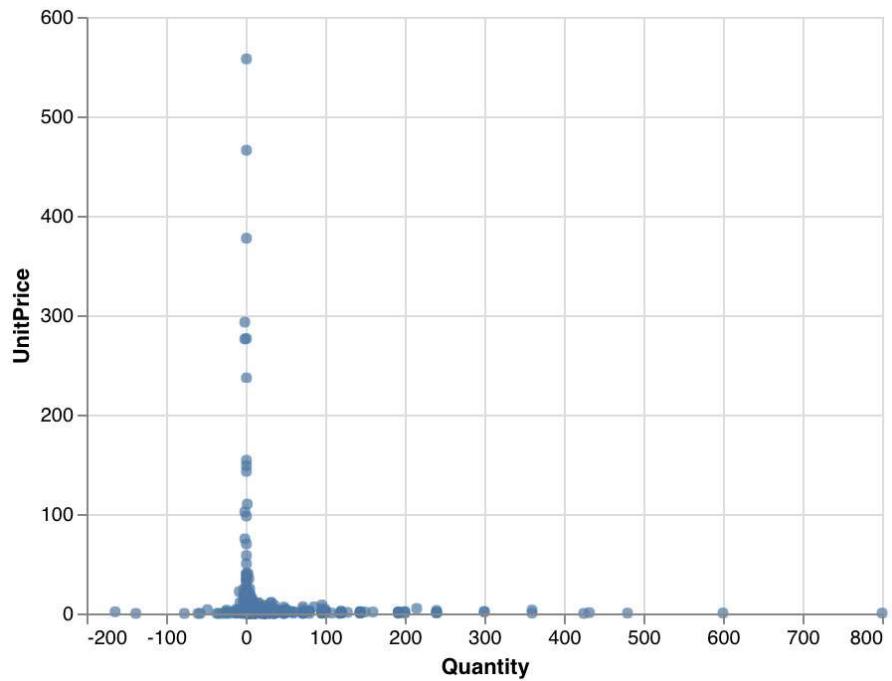


Using the Altair API

- Altair provides the option for combining its methods all together into one single line of code, like this:

```
alt.Chart(sample_df).mark_circle()\n    .encode(x='Quantity',\n            y='UnitPrice')
```

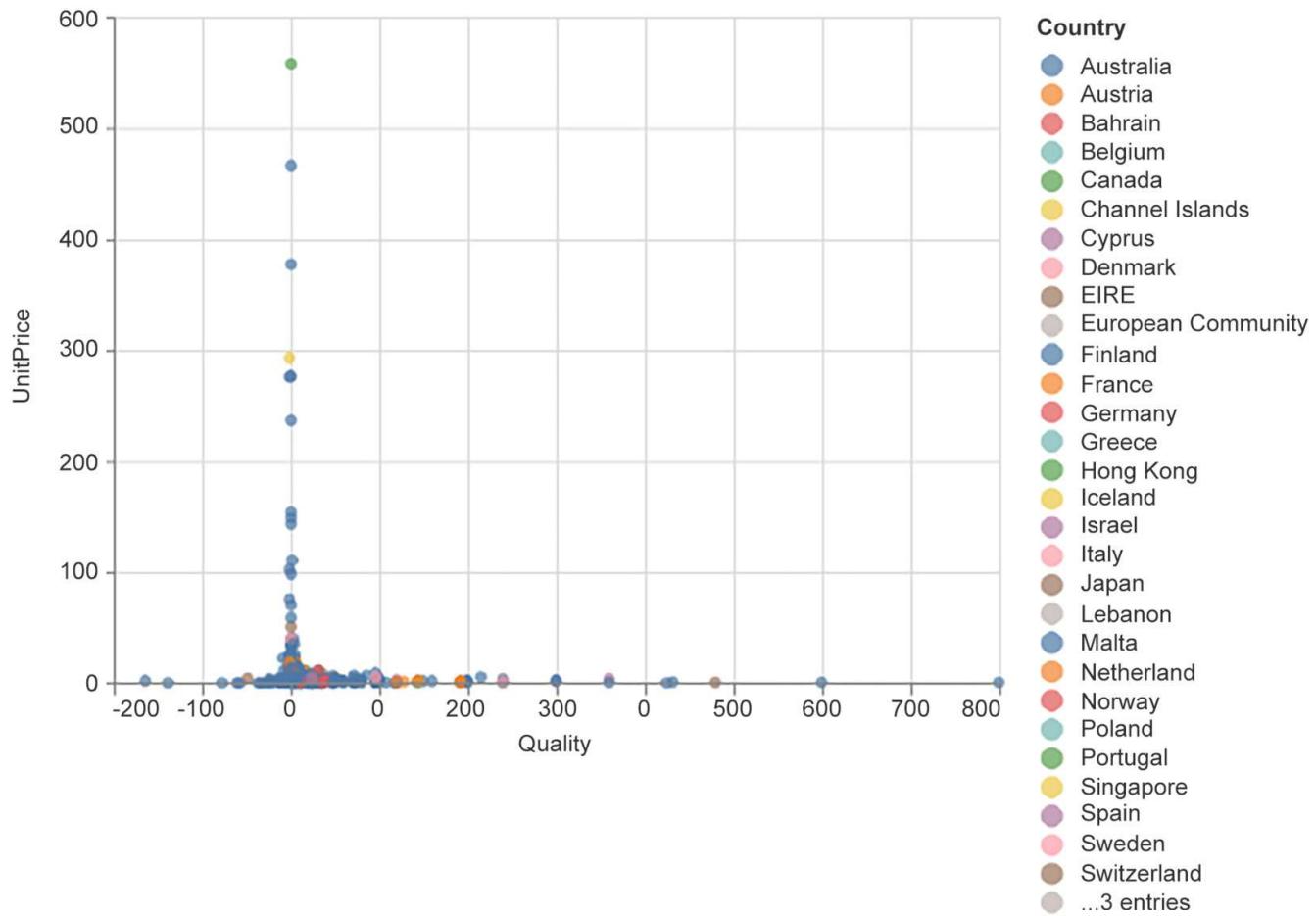
- You should get this output:



Using the Altair API

- Now, let's say we want to visualize the same plot while adding the Country column's information.
- One easy way to do this is to use the color parameter from the encode() method.
- This will color all the data points according to their value in the Country column:

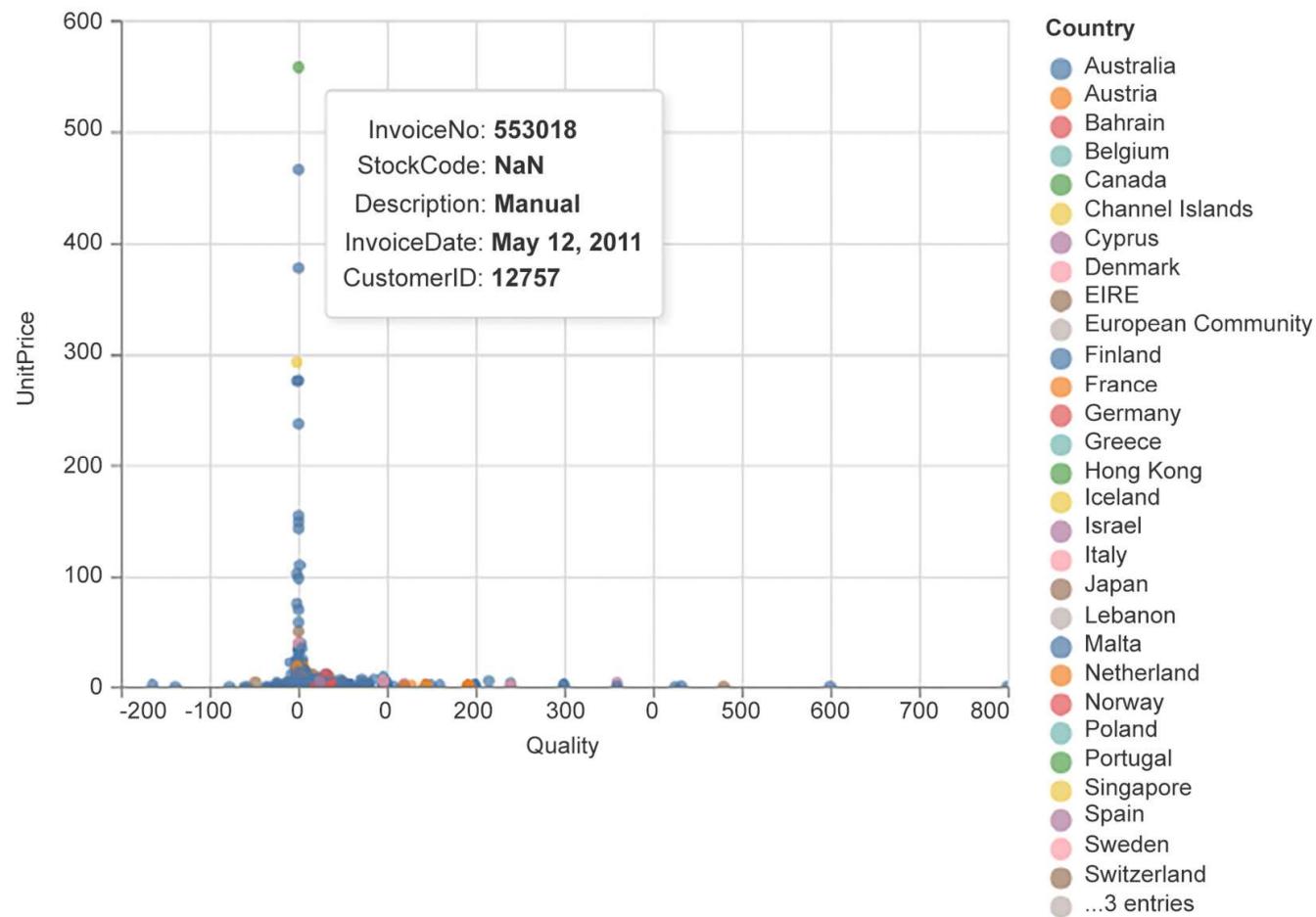
```
alt.Chart(sample_df).mark_circle()\n    .encode(x='Quantity', y='UnitPrice', color='Country')
```



Using the Altair API

- we just need to use the tooltip parameter from the encode() method and specify the list of columns to be displayed and then call the interactive() method to make the whole thing interactive (as seen previously in lesson 5, Performing Your First Cluster Analysis):

```
alt.Chart(sample_df).mark_circle()\n    .encode(x='Quantity', y='UnitPrice', color='Country', \n        tooltip=['InvoiceNo','StockCode','Description',\n            'InvoiceDate','CustomerID']).interactive()
```



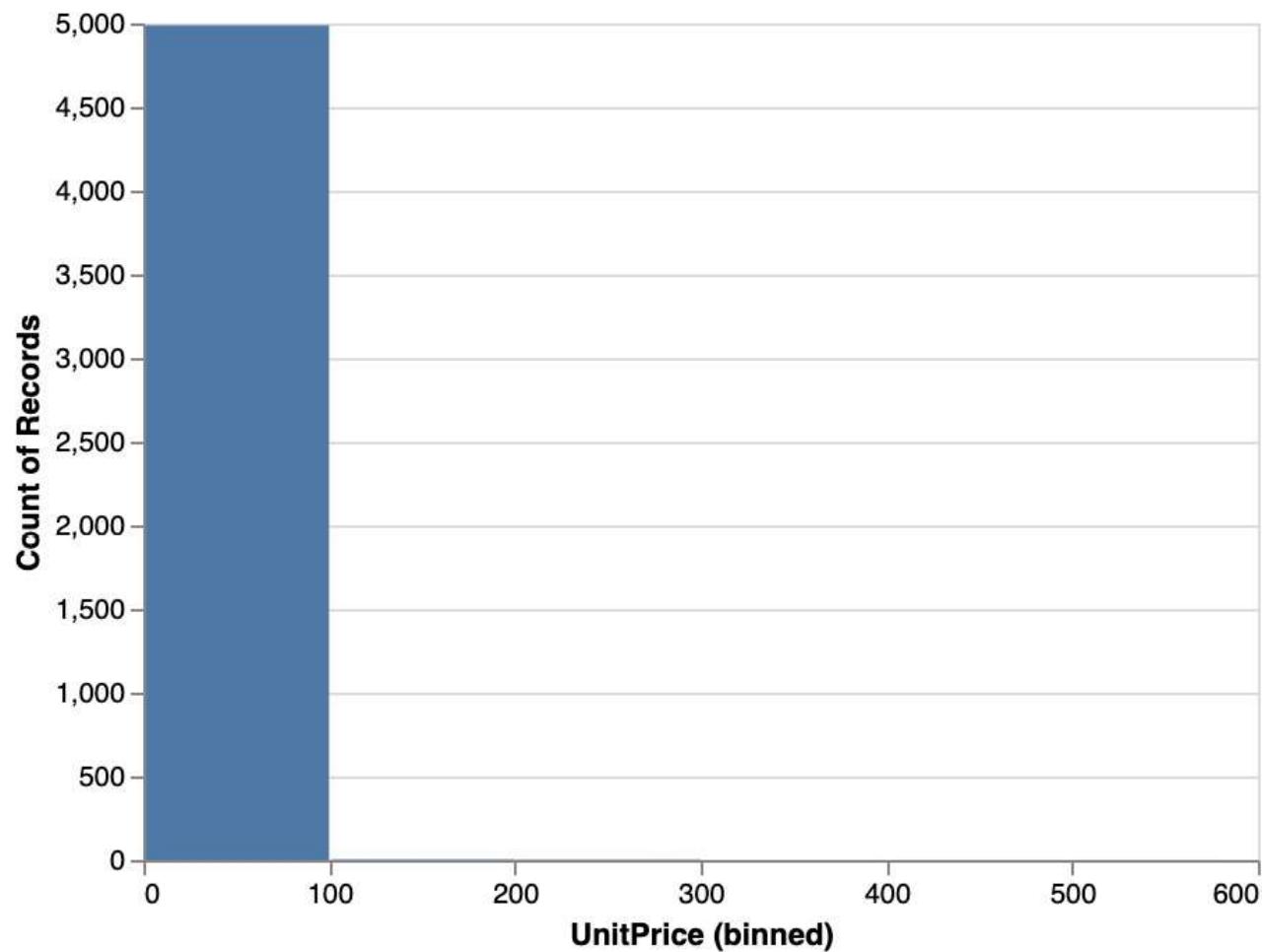
Histogram for Numerical Variables

- Now that we are familiar with the altair API, let's have a look at some specific type of charts that will help us analyze and understand each variable.
- First, let's focus on numerical variables such as UnitPrice or Quantity in the online retail dataset.
- For this type of variable, a histogram is usually used to show the distribution of a given variable.

Histogram for Numerical Variables

- `y='count()'`: This is used for calculating the number of observations and plotting them on the y axis, like so:

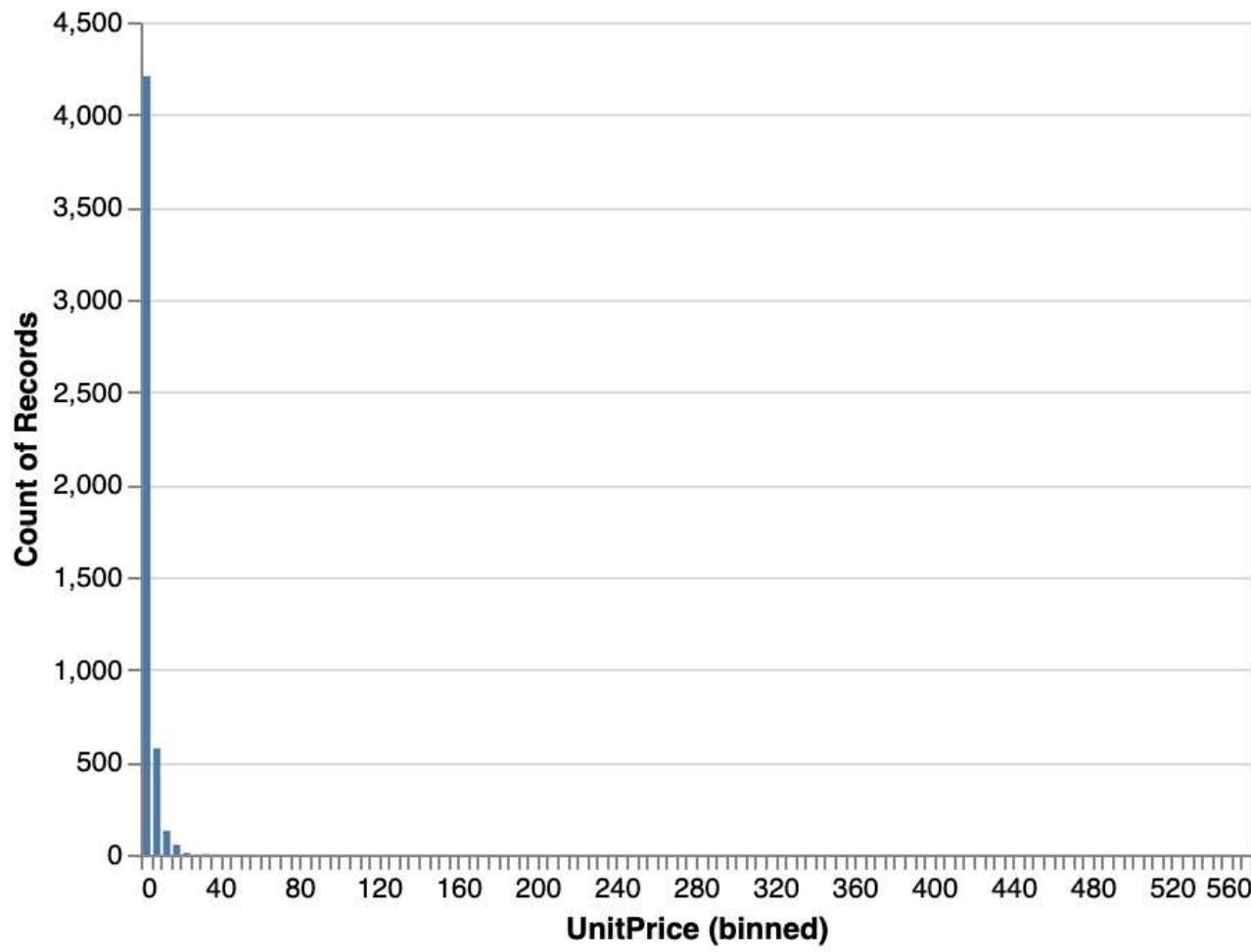
```
alt.Chart(sample_df).mark_bar()\n    .encode(alt.X("UnitPrice:Q", bin=True), \n           y='count())
```



Histogram for Numerical Variables

- With altair, we can specify the values of the parameter bin and we will try this with 5, that is, alt.Bin(step=5):

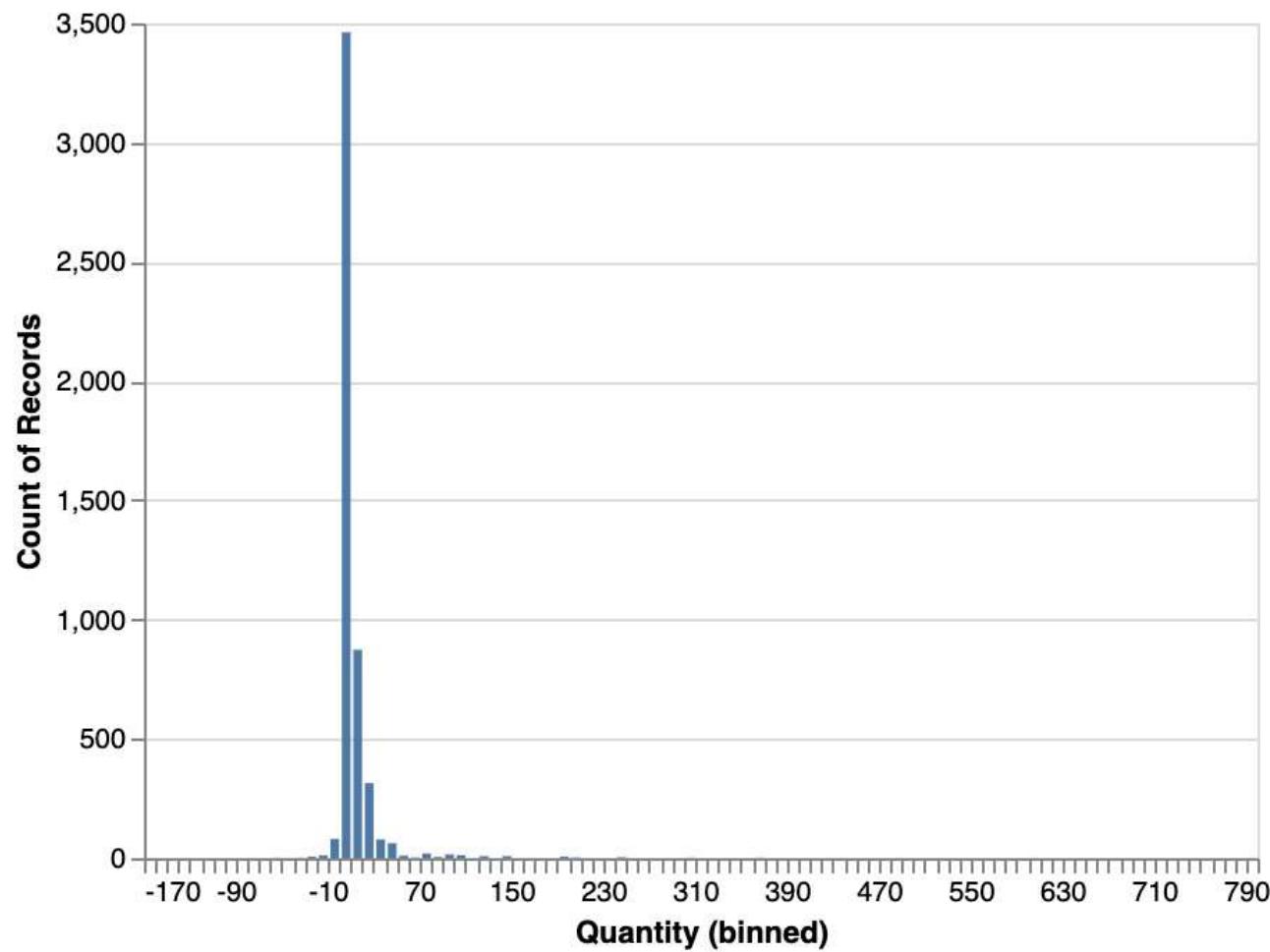
```
alt.Chart(sample_df).mark_bar()\n    .encode(alt.X("UnitPrice:Q", bin=alt.Bin(step=5)), \n           y='count()')
```



Histogram for Numerical Variables

- Let's plot the histogram for the Quantity column with a bin step size of 10:

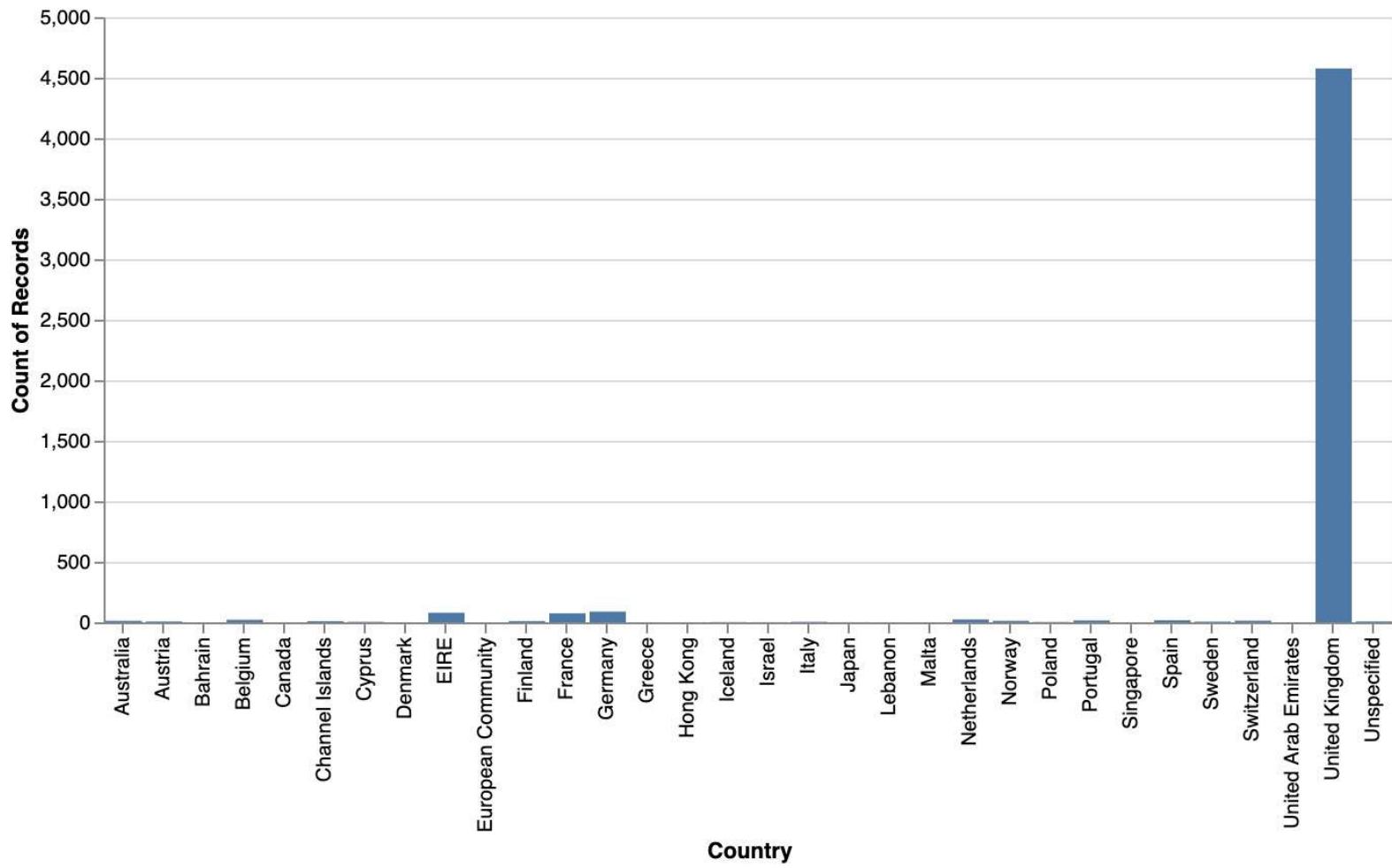
```
alt.Chart(sample_df).mark_bar()\n    .encode(alt.X("Quantity:Q", bin=alt.Bin(step=10)), \n           y='count()')
```



Bar Chart for Categorical Variables

- Let's try this on the Country column and look at the number of records for each of its values:

```
alt.Chart(sample_df).mark_bar()\n    .encode(x='Country',y='count()')
```

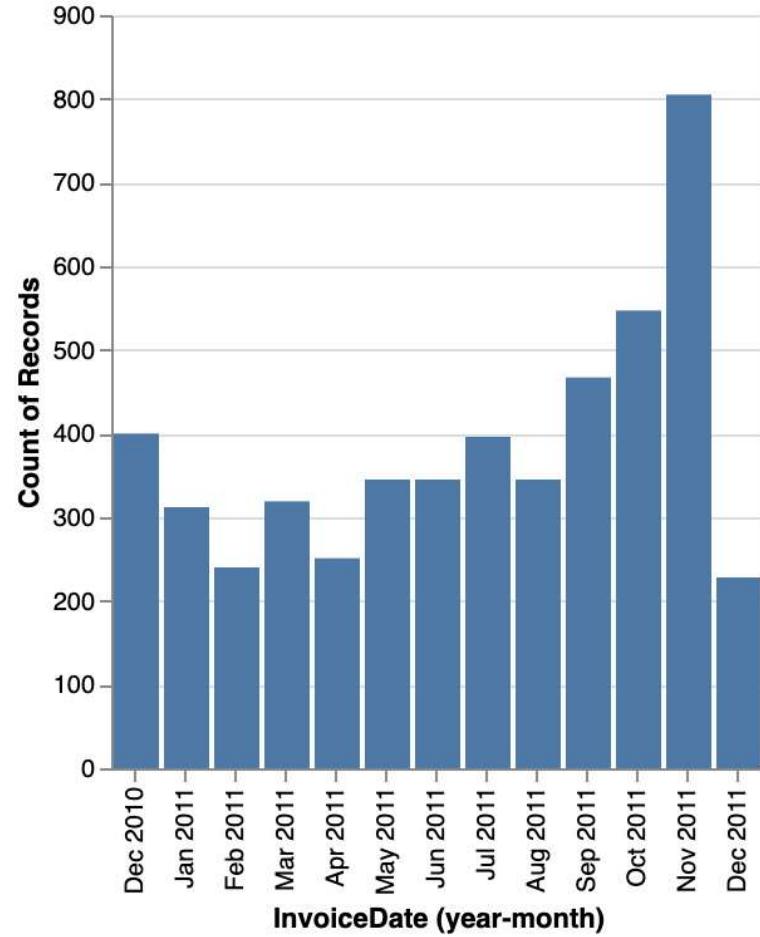


Bar Chart for Categorical Variables

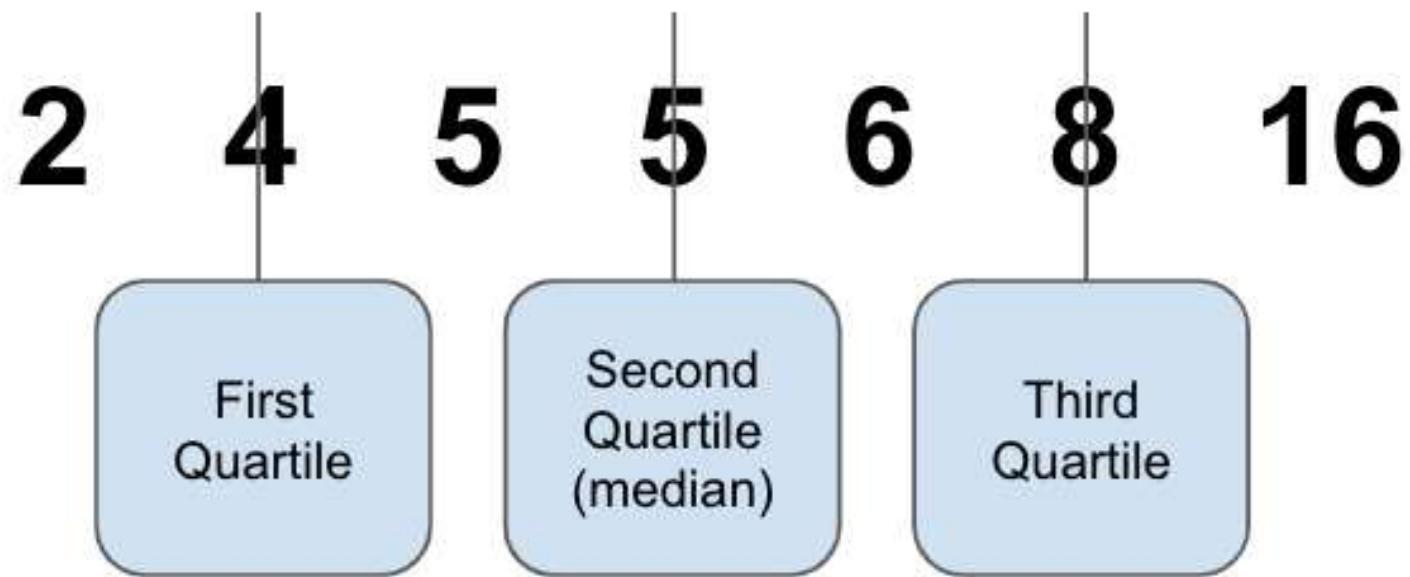
- We also need to specify that the type of this variable is ordinal (there is an order between the values) by adding :O to the column name:

```
alt.Chart(sample_df).mark_bar()\n    .encode(alt.X('yearmonth(InvoiceDate):O'),\n           y='count()')
```

- You should get this output:

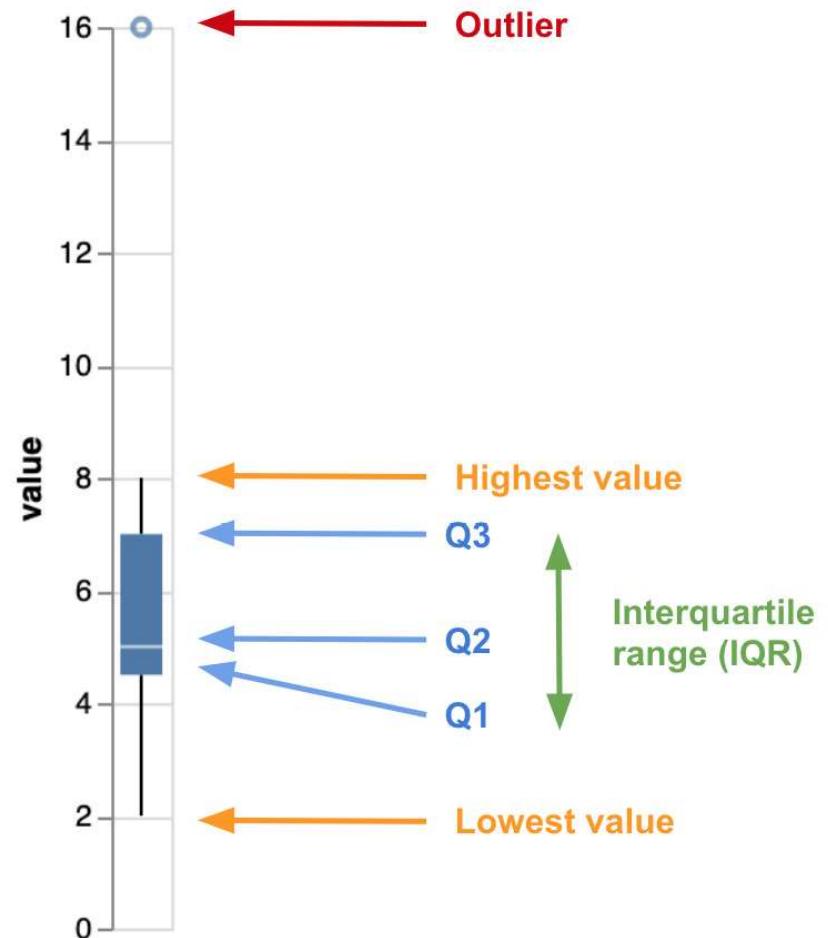


Boxplots



Boxplots

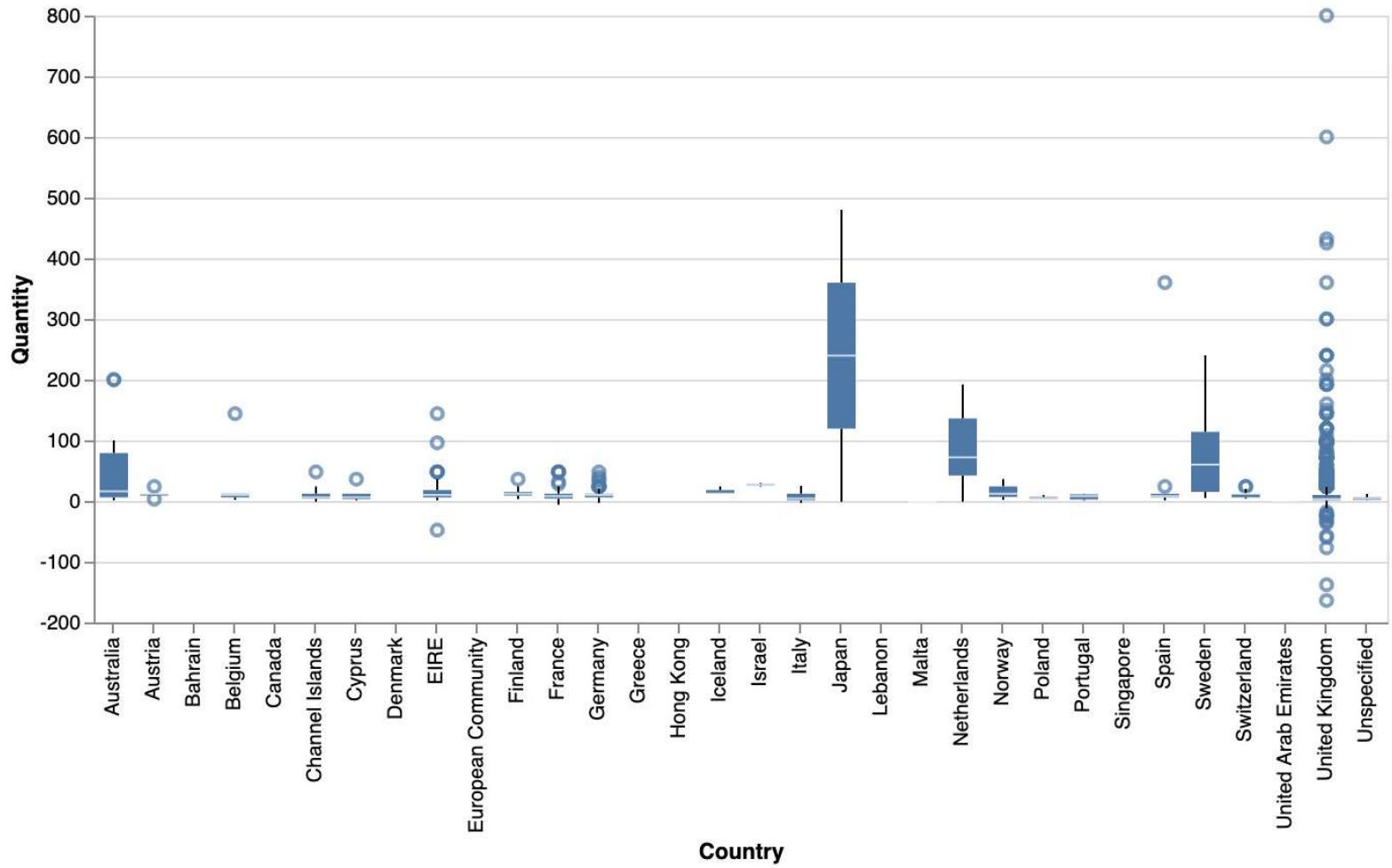
- Outliers, that is, any point outside of the lowest and highest points:



Boxplots

- Another benefit of using a boxplot is to plot the distribution of categorical variables against a numerical variable and compare them.
- Let's try it with the Country and Quantity columns using the `mark_boxplot()` method:

```
alt.Chart(sample_df).mark_boxplot()\n    .encode(x='Country:O', y='Quantity:Q')
```



Complete Exercise 10.04: Visualizing the Ames Housing Dataset with Altair

Complete Activity 10.01: Analyzing Churn Data Using Visual Data Analysis Techniques

Summary

- You just learned a lot regarding how to analyze a dataset.
- This is a very critical step in any data science project.
- Getting a deep understanding of the dataset will help you to better assess the feasibility of achieving the requirements from the business.
- Getting the right data in the right format at the right level of quality is key for getting good predictive performance for any machine learning algorithm.



11. Data Preparation



Overview

- By the end of this lesson you will be able to filter DataFrames with specific conditions
- Remove duplicate or irrelevant records or columns
- Convert variables into different data types
- Replace values in a column and handle missing values and outlier observations.

Introduction

- This lesson will introduce you to the issues that you will encounter frequently during your data scientist career (such as duplicated rows, incorrect data types, incorrect values, and missing values)
- You will learn about the techniques you can use to easily fix them.
- But be careful – some issues that you come across don't necessarily need to be fixed.

Handling Row Duplication

- Start by importing the dataset into a DataFrame:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'data-science/blob/'  
          'master/lesson10/dataset/'  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Handling Row Duplication

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

`df.duplicated()`

Handling Row Duplication

```
0      False
1      False
2      False
3      False
4      False
      ...
541904  False
541905  False
541906  False
541907  False
541908  False
Length: 541909, dtype: bool
```

Handling Row Duplication

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

`df.duplicated().sum()`

- You should get the following output:

`5268`

Handling Row Duplication

- The first API subsets the DataFrame by row or column.
- To filter specific columns, you can provide a list that contains their names.
- For instance, if you want to keep only the variables, that is, InvoiceNo, StockCode, InvoiceDate, and CustomerID, you need to use the following code:

```
df[['InvoiceNo', 'StockCode', 'InvoiceDate', 'CustomerID']]
```

	InvoiceNo	StockCode	InvoiceDate	CustomerID
0	536365	85123A	2010-12-01 08:26:00	17850.0
1	536365	71053	2010-12-01 08:26:00	17850.0
2	536365	84406B	2010-12-01 08:26:00	17850.0
3	536365	84029G	2010-12-01 08:26:00	17850.0
4	536365	84029E	2010-12-01 08:26:00	17850.0
...
541904	581587	22613	2011-12-09 12:50:00	12680.0
541905	581587	22899	2011-12-09 12:50:00	12680.0
541906	581587	23254	2011-12-09 12:50:00	12680.0
541907	581587	23255	2011-12-09 12:50:00	12680.0
541908	581587	22138	2011-12-09 12:50:00	12680.0

541909 rows × 4 columns

Handling Row Duplication

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

```
df[df.duplicated()]
```

Handling Row Duplication

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	2010-12-01 11:45:00	1.25	17908.0	United Kingdom
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	2010-12-01 11:45:00	2.10	17908.0	United Kingdom
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	2010-12-01 11:45:00	2.95	17908.0	United Kingdom
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	2010-12-01 11:45:00	4.95	17908.0	United Kingdom
555	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	2010-12-01 11:49:00	2.95	17920.0	United Kingdom
...
541675	581538	22068	BLACK PIRATE TREASURE CHEST	1	2011-12-09 11:34:00	0.39	14446.0	United Kingdom
541689	581538	23318	BOX OF 6 MINI VINTAGE CRACKERS	1	2011-12-09 11:34:00	2.49	14446.0	United Kingdom
541692	581538	22992	REVOLVER WOODEN RULER	1	2011-12-09 11:34:00	1.95	14446.0	United Kingdom
541699	581538	22694	WICKER STAR	1	2011-12-09 11:34:00	2.10	14446.0	United Kingdom
541701	581538	23343	JUMBO BAG VINTAGE CHRISTMAS	1	2011-12-09 11:34:00	2.08	14446.0	United Kingdom

5268 rows × 8 columns

Handling Row Duplication

- You will use the .loc API to subset the duplicated rows and keep only the selected four columns, as shown in the previous example:

```
df.loc[df.duplicated(), ['InvoiceNo', 'StockCode', \  
                         'InvoiceDate', 'CustomerID']]
```

→

	InvoiceNo	StockCode	InvoiceDate	CustomerID
517	536409	21866	2010-12-01 11:45:00	17908.0
527	536409	22866	2010-12-01 11:45:00	17908.0
537	536409	22900	2010-12-01 11:45:00	17908.0
539	536409	22111	2010-12-01 11:45:00	17908.0
555	536412	22327	2010-12-01 11:49:00	17920.0
...
541675	581538	22068	2011-12-09 11:34:00	14446.0
541689	581538	23318	2011-12-09 11:34:00	14446.0
541692	581538	22992	2011-12-09 11:34:00	14446.0
541699	581538	22694	2011-12-09 11:34:00	14446.0
541701	581538	23343	2011-12-09 11:34:00	14446.0

5268 rows × 4 columns

Handling Row Duplication

- You can change this behavior by specifying the `keep` parameter.
- If you want to keep the last duplicate, you need to specify `keep='last'`:

```
df.loc[df.duplicated(keep='last'), ['InvoiceNo', 'StockCode', \
    'InvoiceDate', 'CustomerID']]
```

→

	InvoiceNo	StockCode	InvoiceDate	CustomerID
485	536409	22111	2010-12-01 11:45:00	17908.0
489	536409	22866	2010-12-01 11:45:00	17908.0
494	536409	21866	2010-12-01 11:45:00	17908.0
521	536409	22900	2010-12-01 11:45:00	17908.0
548	536412	22327	2010-12-01 11:49:00	17920.0
...
541640	581538	22992	2011-12-09 11:34:00	14446.0
541644	581538	22694	2011-12-09 11:34:00	14446.0
541646	581538	23275	2011-12-09 11:34:00	14446.0
541656	581538	23318	2011-12-09 11:34:00	14446.0
541666	581538	23343	2011-12-09 11:34:00	14446.0

5268 rows × 4 columns

Handling Row Duplication

- If you want to mark all the duplicate records as duplicates, you will have to use keep=False:

```
df.loc[df.duplicated(keep=False), ['InvoiceNo', 'StockCode',\n    'InvoiceDate', 'CustomerID']]
```



InvoiceNo StockCode

InvoiceDate

CustomerID

485	536409	22111	2010-12-01 11:45:00	17908.0
-----	--------	-------	---------------------	---------

489	536409	22866	2010-12-01 11:45:00	17908.0
-----	--------	-------	---------------------	---------

494	536409	21866	2010-12-01 11:45:00	17908.0
-----	--------	-------	---------------------	---------

517	536409	21866	2010-12-01 11:45:00	17908.0
-----	--------	-------	---------------------	---------

521	536409	22900	2010-12-01 11:45:00	17908.0
-----	--------	-------	---------------------	---------

...
-----	-----	-----	-----	-----

541675	581538	22068	2011-12-09 11:34:00	14446.0
--------	--------	-------	---------------------	---------

541689	581538	23318	2011-12-09 11:34:00	14446.0
--------	--------	-------	---------------------	---------

541692	581538	22992	2011-12-09 11:34:00	14446.0
--------	--------	-------	---------------------	---------

541699	581538	22694	2011-12-09 11:34:00	14446.0
--------	--------	-------	---------------------	---------

541701	581538	23343	2011-12-09 11:34:00	14446.0
--------	--------	-------	---------------------	---------

10147 rows × 4 columns

Handling Row Duplication

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

Handling Row Duplication

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom

- The output of this method is a new DataFrame that contains unique records where only the first occurrence of duplicates has been kept.

Handling Row Duplication

- Let's see how many duplicate rows there are while only looking at the InvoiceNo, StockCode, invoiceDate, and CustomerID columns:

```
df.duplicated(subset=['InvoiceNo', 'StockCode', 'InvoiceDate',  
'CustomerID'], keep='first').sum()
```

- You should get the following output:

10677

Complete Exercise 11.01: Handling Duplicates in a Breast Cancer Dataset

Converting Data Types

- First, you must import pandas:
`import pandas as pd`
- Then, you need to assign the URL to the dataset to a new variable:

```
file_url = 'https://github.com/fenago/\  
    'data-science/blob/\  
    'master/lesson10/dataset/\  
    'Online%20Retail.xlsx?raw=true'
```

Converting Data Types

- Let's load the dataset into a pandas DataFrame using `read_excel()`:

```
df = pd.read_excel(file_url)
```

- Finally, let's print the data type of each column:

```
df.dtypes
```

Converting Data Types

- You should get the following output:

```
InvoiceNo          object
StockCode          object
Description        object
Quantity           int64
InvoiceDate        datetime64[ns]
UnitPrice          float64
CustomerID         float64
Country            object
dtype: object
```

Converting Data Types

- Here, we are going to change its type to object:

```
df = pd.read_excel(file_url, dtype={'CustomerID': 'category'})  
df.dtypes
```

- You should get the following output:

InvoiceNo	object
StockCode	object
Description	object
Quantity	int64
InvoiceDate	datetime64[ns]
UnitPrice	float64
CustomerID	category
Country	object
dtype:	object

Converting Data Types

- For instance, if you want to change the InvoiceNo column into a categorical variable, you would do the following:

```
df['InvoiceNo'] = df['InvoiceNo'].astype('category')  
df.dtypes
```

- You should get the following output:

InvoiceNo	category
StockCode	object
Description	object
Quantity	int64
InvoiceDate	datetime64[ns]
UnitPrice	float64
CustomerID	category
Country	object
dtype:	object

Converting Data Types

- Once these have been changed into categorical variables, pandas will automatically list all the values.
- They can be accessed using the .cat.categories attribute:
`df['InvoiceNo'].cat.categories`
- You should get the following output:

```
Index([ 536365,      536366,      536367,      536368,      536369,      536370,
        536371,      536372,      536373,      536374,
        ...
       'C581464', 'C581465', 'C581466', 'C581468', 'C581470', 'C581484',
       'C581490', 'C581499', 'C581568', 'C581569'],
       dtype='object', length=25900)
```

Complete Exercise 11.02: Converting Data Types for the Ames Housing Dataset

Handling Incorrect Values

- Let's learn how to detect such issues in real life by using the Online Retail dataset.
- First, you need to load the data into a pandas DataFrame:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
           'data-science/blob/'  
           'master/lesson10/dataset/'  
           'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Handling Incorrect Values

- In this dataset, there are two variables that seem to be related to each other: StockCode and Description.
- The first one contains the identifier code of the items sold and the other one contains their descriptions.
- However, if you look at some of the examples, such as StockCode 23131, the Description column has different values:

```
df.loc[df['StockCode'] == 23131, 'Description'].unique()
```

- You should get the following output

```
array(['MISTLETOE HEART WREATH CREAM', 'MISELTOE HEART WREATH WHITE',
       'MISELTOE HEART WREATH CREAM', '?', 'had been put aside', nan],
      dtype=object)
```

Handling Incorrect Values

- Let's focus on the misspelling issue. What we need to do here is modify the incorrect spelling and replace it with the correct value.
- First, let's create a new column called StockCodeDescription, which is an exact copy of the Description column:

```
df['StockCodeDescription'] = df['Description']
```

Handling Incorrect Values

- You need to use .loc and filter the rows and columns you want, that is, all rows with StockCode == 21131 and Description == MISELTOE HEART WREATH CREAM and the Description column:

```
df.loc[(df['StockCode'] == 23131) \
& (df['StockCodeDescription'] \
== 'MISELTOE HEART WREATH CREAM'), \
'StockCodeDescription'] = 'MISTLETOE HEART \
WREATH CREAM'
```

Handling Incorrect Values

- If you reprint the value for this issue, you will see that the misspelling value has been fixed and is not present anymore:
`df.loc[df['StockCode'] == 23131, 'StockCodeDescription'].unique()`
- You should get the following output:

```
array(['MISTLETOE HEART WREATH CREAM', 'MISELTOE HEART WREATH WHITE', '?',
       'had been put aside', nan], dtype=object)
```

Handling Incorrect Values

- You will also have to subset the rows that don't have missing values for the Description column.
- This can be done by providing the na=False parameter to the .str.contains() method:
`df.loc[df['StockCodeDescription']\n .str.contains('MISEL', na=False),]`
- You should get the following output:

InvoiceNo	StockCode	Description
186760	552882	23131 MISELTOE HEART WREATH WHITE
186761	552882	23130 MISELTOE HEART WREATH
195286	553711	23130 MISELTOE HEART WREATH
195288	553711	23131 MISELTOE HEART WREATH WHITE
372887	569252	23130 MISELTOE HEART WREATH
373325	569324	23131 MISELTOE HEART WREATH WHITE
373327	569324	23130 MISELTOE HEART WREATH
377632	569558	23131 MISELTOE HEART WREATH WHITE
377635	569558	23130 MISELTOE HEART WREATH

Handling Incorrect Values

- This method takes the string of characters to be replaced and the replacement string as parameters:

```
df['StockCodeDescription'] = df['StockCodeDescription']\n    .str.replace\\n        ('MISELTOE', 'MISTLETOE')
```

Handling Incorrect Values

- Now, if you print all the rows that contain the misspelling of MISEL, you will see that no such rows exist anymore:

```
df.loc[df['StockCodeDescription']\n      .str.contains('MISEL', na=False),]
```

- You should get the following output

```
InvoiceNo StockCode Description Quantity InvoiceDate UnitPrice CustomerID Country StockCodeDescription
```

Complete Exercise 11.03: Fixing Incorrect Values in the State Column

Handling Missing Values

- Let's see it in action on the Online Retail dataset. First, you need to import pandas and load the data into a DataFrame:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'data-science/blob/'  
          'master/lesson10/dataset/'  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Handling Missing Values

- The `.isna()` method returns a pandas series with a binary value for each cell of a DataFrame and states whether it is missing a value (True) or not (False):

`df.isna()`

- You should get the following output:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False

Handling Missing Values

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

```
df.isna().sum()
```

- You should get the following output:

```
InvoiceNo          0
StockCode          0
Description       1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID      135080
Country            0
dtype: int64
```

Handling Missing Values

- You can use the output of the `.isna()` method to subset the rows with missing values:

```
df[df['Description'].isna()]
```

- You should get the following output:

InvoiceNo	StockCode	Description	Quantity		InvoiceDate	UnitPrice	CustomerID	Country
622	536414	22139	NaN	56	2010-12-01 11:52:00	0.0	NaN	United Kingdom
1970	536545	21134	NaN	1	2010-12-01 14:32:00	0.0	NaN	United Kingdom
1971	536546	22145	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1972	536547	37509	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1987	536549	85226A	NaN	1	2010-12-01 14:34:00	0.0	NaN	United Kingdom
1988	536550	85044	NaN	1	2010-12-01 14:34:00	0.0	NaN	United Kingdom

Handling Missing Values

- By default, it will look at all the columns. You can specify a list of columns for it to look for with the subset parameter:

```
df.dropna(subset=['Description'])
```

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

```
df.dropna(subset=['Description'], inplace=True)
```

Handling Missing Values

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

`df.isna().sum()`

- You should get the following output:

```
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID        133626
Country            0
dtype: int64
```

Handling Missing Values

- As you can see, there are no more missing values in the Description column.
- Let's have a look at the CustomerID column:
`df[df['CustomerID'].isna()]`
- You should get the following output:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1443	536544	21773 DECORATIVE ROSE BATHROOM BOTTLE	1	2010-12-01 14:32:00	2.51	NaN	United Kingdom
1444	536544	21774 DECORATIVE CATS BATHROOM BOTTLE	2	2010-12-01 14:32:00	2.51	NaN	United Kingdom
1445	536544	21786 POLKA DOT RAIN HAT	4	2010-12-01 14:32:00	0.85	NaN	United Kingdom
1446	536544	21787 RAIN PONCHO RETROSPOT	2	2010-12-01 14:32:00	1.66	NaN	United Kingdom
1447	536544	21790 VINTAGE SNAP CARDS	9	2010-12-01 14:32:00	1.66	NaN	United Kingdom
...
541536	581498	85099B JUMBO BAG RED RETROSPOT	5	2011-12-09 10:26:00	4.13	NaN	United Kingdom
541537	581498	85099C JUMBO BAG BAROQUE BLACK WHITE	4	2011-12-09 10:26:00	4.13	NaN	United Kingdom
541538	581498	85150 LADIES & GENTLEMEN METAL SIGN	1	2011-12-09 10:26:00	4.96	NaN	United Kingdom
541539	581498	85174 S/4 CACTI CANDLES	1	2011-12-09 10:26:00	10.79	NaN	United Kingdom
541540	581498	DOT DOTCOM POSTAGE	1	2011-12-09 10:26:00	1714.17	NaN	United Kingdom

133626 rows × 8 columns

Handling Missing Values

- Provide the value to be imputed as Missing and use `inplace=True` as a parameter:

```
df['CustomerID'].fillna('Missing', inplace=True)
```

```
df[1443:1448]
```

- You should get the following output:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
1444	536544	21774 DECORATIVE CATS BATHROOM BOTTLE	2	2010-12-01 14:32:00	2.51	Missing
1445	536544	21786 POLKADOT RAIN HAT	4	2010-12-01 14:32:00	0.85	Missing
1446	536544	21787 RAIN PONCHO RETROSPOT	2	2010-12-01 14:32:00	1.66	Missing
1447	536544	21790 VINTAGE SNAP CARDS	9	2010-12-01 14:32:00	1.66	Missing
1448	536544	21791 VINTAGE HEADS AND TAILS CARD GAME	2	2010-12-01 14:32:00	2.51	Missing

Handling Missing Values

- Let's see if we have any missing values in the dataset:
`df.isna().sum()`
- You should get this output:

```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID    0
Country        0
dtype: int64
```

Complete Exercise 11.04: Fixing Missing Values for the Horse Colic Dataset

Complete Activity 11.01: Preparing the Speed Dating Dataset

Summary

- In this lesson, you learned how important it is to prepare any given dataset and fix the main quality issues it has.
- This is critical because the cleaner a dataset is, the easier it will be for any machine learning model to easily learn about the relevant patterns.
- On top of this, most algorithms can't handle issues such as missing values, so they must be handled prior to the modeling phase.



12. Feature Engineering



Overview

- By the end of this lesson, you will be able to merge multiple datasets together; bin categorical and numerical variables; perform aggregation on data; and manipulate dates using pandas.

Introduction

- The objective of feature engineering is to provide more information for the analysis you are performing on or the machine learning algorithms you will train on.
- Adding more information will help you to achieve better and more accurate results.
- New features can come from internal data sources such as another table from databases or from different systems.

Merging Datasets

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

Merging Datasets

- First, we need to import the Online Retail dataset into a pandas DataFrame:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'data-science/blob/'  
          'master/lesson12/Dataset/'  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)  
df.head()
```

Merging Datasets

- You should get the following output.

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Merging Datasets

- Let's focus on 2010 first:

```
uk_holidays_2010 = pd.read_csv(\n    'https://date.nager.at/PublicHoliday/\\'\n    'Country/GB/2010/CSV')
```

- We can print its shape to see how many rows and columns it has:

```
uk_holidays_2010.shape
```

- You should get the following output.

```
(13, 8)
```

Merging Datasets

- Let's print the first five rows of this DataFrame:
`uk_holidays_2010.head()`
- You should get the following output:

	Date	LocalName	Name	CountryCode	Fixed	Global	LaunchYear	Type
0	2010-01-01	New Year's Day	New Year's Day	GB	False	True	NaN	Public
1	2010-01-04	New Year's Day	New Year's Day	GB	False	False	NaN	Public
2	2010-03-17	Saint Patrick's Day	Saint Patrick's Day	GB	True	False	NaN	Public
3	2010-04-02	Good Friday	Good Friday	GB	False	True	NaN	Public
4	2010-04-05	Easter Monday	Easter Monday	GB	False	False	NaN	Public

Merging Datasets

- Now that we have the list of public holidays for 2010, let's extract the ones for 2011:

```
uk_holidays_2011 = pd.read_csv\  
    ('https://date.nager.at/PublicHoliday'\  
     'Country/GB/2011/CSV')
```

```
uk_holidays_2011.shape
```

- You should get the following output.
(15, 8)

Merging Datasets

- There were 15 public holidays in 2011. Now we need to combine the records of these two DataFrames.
- We will use the `.append()` method from pandas and assign the results into a new DataFrame:

```
uk_holidays =  
uk_holidays_2010.append(uk_holidays_2011)
```

Merging Datasets

- Let's check we have the right number of rows after appending the two DataFrames:
`uk_holidays.shape`
- You should get the following output:
`(28, 8)`

Merging Datasets

- For example, the date 2010-12-01 08:26:00 will first be converted into a string and then we will keep only the first 10 characters, which will be 2010-12-01.
- We are going to save these results into a new column called InvoiceDay:

```
df['InvoiceDay'] = df['InvoiceDate'].astype(str)\n                .str.slice(stop=10)
```

```
df.head()
```

Merging Datasets

- The output is as follows:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceDay
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01

The Left Join

- The left join will keep all the rows from the first DataFrame, which is the Online Retail dataset (the left-hand side) and join it to the matching rows from the second DataFrame, which is the UK Public Holidays dataset (the right-hand side), as shown



The Left Join

- These parameters are clubbed together as shown in the following code snippet:

```
df_left = pd.merge(df, uk_holidays, left_on='InvoiceDay', \
                    right_on='Date', how='left')  
df_left.shape
```

- You should get the following output:
(541909, 17)

The Left Join

- We got the exact same number of rows as the original Online Retail DataFrame, which is expected for a left join.
- Let's have a look at the first five rows:

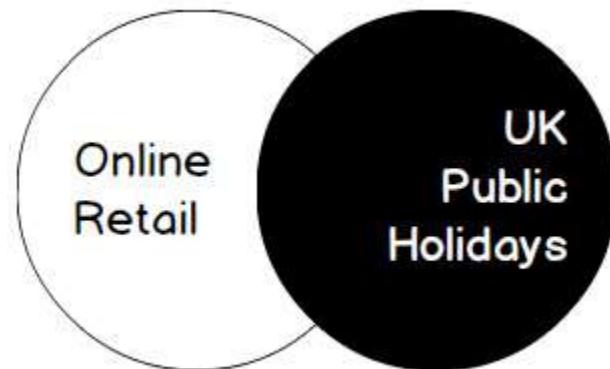
```
df_left.head()
```

The Left Join

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceDay	Date	LocalName	Name	CountryCode	Fixed
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	NaN	NaN	NaN	NaN	NaN
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	NaN	NaN	NaN	NaN	NaN
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01	NaN	NaN	NaN	NaN	NaN
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	NaN	NaN	NaN	NaN	NaN
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	NaN	NaN	NaN	NaN	NaN

The Right Join

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



The Right Join

- These parameters are clubbed together as shown in the following code snippet:

```
df_right = df.merge(uk_holidays, left_on='InvoiceDay', \
                    right_on='Date', how='right')
```

```
df_right.shape
```

- You should get the following output:

```
(9602, 17)
```

The Right Join

- For instance, looking at the first rows of the merged DataFrame, we can see there were multiple purchases on January 4, 2011, so all of them have been matched with the corresponding public holiday.
- Have a look at the following code snippet:

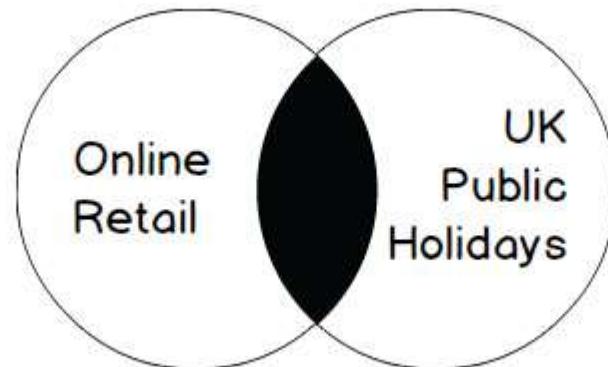
```
df_right.head()
```

The Right Join

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceDay	Date	LocalName	Name	CountryCode	Fixed
0	539993	22386	JUMBO BAG PINK POLKADOT	10.0	2011-01-04 10:00:00	1.95	13313.0	United Kingdom	2011-01-04	2011-01-04	New Year's Day	New Year's Day	GB	False
1	539993	21499	BLUE POLKADOT WRAP	25.0	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011-01-04	2011-01-04	New Year's Day	New Year's Day	GB	False
2	539993	21498	RED RETROSPOT WRAP	25.0	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011-01-04	2011-01-04	New Year's Day	New Year's Day	GB	False
3	539993	22379	RECYCLING BAG RETROSPOT	5.0	2011-01-04 10:00:00	2.10	13313.0	United Kingdom	2011-01-04	2011-01-04	New Year's Day	New Year's Day	GB	False
4	539993	20718	RED RETROSPOT SHOPPER BAG	10.0	2011-01-04 10:00:00	1.25	13313.0	United Kingdom	2011-01-04	2011-01-04	New Year's Day	New Year's Day	GB	False

The Right Join

- There are two other types of merging: inner and outer.
- An inner join will only keep the rows that match between the two tables:



The Right Join

- These parameters are clubbed together as shown in the following code snippet:

```
df_inner = df.merge(uk_holidays, left_on='InvoiceDay', \
                    right_on='Date', how='inner')
```

```
df_inner.shape
```

- You should get the following output:

```
(9579, 17)
```

The Right Join

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



The Right Join

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

```
df_outer = df.merge(uk_holidays, left_on='InvoiceDay', \
                    right_on='Date', how='outer')
```

```
df_outer.shape
```

- You should get the following output:

```
(541932, 17)
```

Complete Exercise 12.01: Merging the ATO Dataset with the Postcode Data

Binning Variables

- One possible solution is to not use these columns and drop them, but in that case, you may lose some very important and critical information for the business.
- Another solution is to create a more consolidated version of these columns by reducing the number of unique values to a smaller number, let's say 100.
- This would drastically speed up the training process for the algorithm without losing too much information.

Binning Variables

- Let's see how we can achieve this on the Online Retail dataset.
- First, we need to load the data:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
          'The-Data-Science-Workshop/blob/'  
          'master/lesson12/Dataset/'  
          'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Binning Variables

- In lesson 10, Analyzing a Dataset we learned that the Country column contains 38 different unique values:
`df['Country'].unique()`
- You should get the following output:

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

Binning Variables

- First, let's create a new column called Country_bin by copying the Country column:
`df['Country_bin'] = df['Country']`
- Then, we are going to create a list called asian_countries containing the name of Asian countries from the list of unique values for the Country column:
`asian_countries = ['Japan', 'Hong Kong', 'Singapore']`

Binning Variables

- And finally, using the `.loc()` and `.isin()` methods from pandas, we are going to change the value of `Country_bin` to Asia for all of the countries that are present in the `asian_countries` list:

```
df.loc[df['Country'].isin(asian_countries), \  
       'Country_bin'] = 'Asia'
```

Binning Variables

- Now, if we print the list of unique values for this new column, we will see the three Asian countries (Japan, Hong Kong, and Singapore) have been replaced by the value Asia:

```
df['Country_bin'].unique()
```

- You should get the following output:

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Asia', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Lebanon',
       'United Arab Emirates', 'Saudi Arabia', 'Czech Republic', 'Canada',
       'Unspecified', 'Brazil', 'USA', 'European Community', 'Malta',
       'RSA'], dtype=object)
```

Binning Variables

- Let's perform the same process for Middle Eastern countries:

```
m_east_countries = ['Israel', 'Bahrain', 'Lebanon', \
                     'United Arab Emirates', 'Saudi Arabia']
df.loc[df['Country'].isin(m_east_countries), \
       'Country_bin'] = 'Middle East'
df['Country_bin'].unique()
```

Binning Variables

- You should get the following output:

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Asia', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Middle East', 'Finland', 'Greece', 'Czech Republic', 'Canada',
       'Unspecified', 'Brazil', 'USA', 'European Community', 'Malta',
       'RSA'], dtype=object)
```

Binning Variables

- Finally, let's group all countries from North and South America together:

```
american_countries = ['Canada', 'Brazil', 'USA']
df.loc[df['Country'].isin(american_countries), \
       'Country_bin'] = 'America'
df['Country_bin'].unique()
```

- You should get the following output:

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Asia', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Middle East', 'Finland', 'Greece', 'Czech Republic', 'America',
       'Unspecified', 'European Community', 'Malta', 'RSA'], dtype=object)
```

Binning Variables

```
df['Country_bin'].nunique()
```

- You should get the following output:

30

- 30 is the number of unique values for the Country_bin column.
- So we reduced the number of unique values in this column from 38 to 30.

Complete Exercise 12.02: Binning the YearBuilt Variable from the AMES Housing Dataset

Manipulating Dates

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

Manipulating Dates

- Let's have a look again at the type of each column from the Online Retail dataset:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
    'The-Data-Science-Workshop/blob/'  
    'master/lesson12/Dataset/'  
    'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)  
df.dtypes
```

Manipulating Dates

- You should get the following output:

```
InvoiceNo          object
StockCode          object
Description        object
Quantity           int64
InvoiceDate        datetime64[ns]
UnitPrice          float64
CustomerID         float64
Country            object
dtype: object
```

Manipulating Dates

- In this case, you will have to manually convert them using the `.to_datetime()` method:

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

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

```
df['InvoiceDate'].dt.year
```

Manipulating Dates

- You should get the following output:

```
→ 0      2010
    1      2010
    2      2010
    3      2010
    4      2010
          ...
541904  2011
541905  2011
541906  2011
541907  2011
541908  2011
Name: InvoiceDate, Length: 541909, dtype: int64
```

Manipulating Dates

- You can get the day of the week from a date using the `.dt.dayofweek` attribute:

```
df['InvoiceDate'].dt.dayofweek
```

- You should get the following output.

```
0      2  
1      2  
2      2  
3      2  
4      2  
..  
541904    4  
541905    4  
541906    4  
541907    4  
541908    4  
Name: InvoiceDate, Length: 541909, dtype: int64
```

Manipulating Dates

- We can, for instance, add 3 days to each date by using pandas time-series offset object, `pd.tseries.offsets.Day(3)`:

`df['InvoiceDate'] + pd.tseries.offsets.Day(3)`

- You should get the following output:

```
0      2010-12-04 08:26:00
1      2010-12-04 08:26:00
2      2010-12-04 08:26:00
3      2010-12-04 08:26:00
4      2010-12-04 08:26:00
...
541904 2011-12-12 12:50:00
541905 2011-12-12 12:50:00
541906 2011-12-12 12:50:00
541907 2011-12-12 12:50:00
541908 2011-12-12 12:50:00
Name: InvoiceDate, Length: 541909, dtype: datetime64[ns]
```

Manipulating Dates

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

```
df['InvoiceDate'] + pd.tseries.offsets.BusinessDay(-1)
```

- You should get the following output:

```
0      2010-11-30 08:26:00  
1      2010-11-30 08:26:00  
2      2010-11-30 08:26:00  
3      2010-11-30 08:26:00  
4      2010-11-30 08:26:00  
...  
541904 2011-12-08 12:50:00  
541905 2011-12-08 12:50:00  
541906 2011-12-08 12:50:00  
541907 2011-12-08 12:50:00  
541908 2011-12-08 12:50:00  
Name: InvoiceDate, Length: 541909, dtype: datetime64[ns]
```

Manipulating Dates

- For instance, if you want to get the first day of the month from a date, you do:

```
df['InvoiceDate'] + pd.Timedelta(1, unit='MS')
```

- You should get the following output:

```
0      2010-12-01 08:26:00.001
1      2010-12-01 08:26:00.001
2      2010-12-01 08:26:00.001
3      2010-12-01 08:26:00.001
4      2010-12-01 08:26:00.001
...
541904 2011-12-09 12:50:00.001
541905 2011-12-09 12:50:00.001
541906 2011-12-09 12:50:00.001
541907 2011-12-09 12:50:00.001
541908 2011-12-09 12:50:00.001
Name: InvoiceDate, Length: 541909, dtype: datetime64[ns]
```

Complete Exercise 12.03: Date Manipulation on Financial Services Consumer Complaints

Performing Data Aggregation

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

Performing Data Aggregation

- Let's see how to do it on the Online Retail dataset.
First, we need to import the data:

```
import pandas as pd  
file_url = 'https://github.com/fenago/'  
    'The-Data-Science-Workshop/blob/'  
    'master/lesson12/Dataset/'  
    'Online%20Retail.xlsx?raw=true'  
df = pd.read_excel(file_url)
```

Performing Data Aggregation

- Let's calculate the total quantity of items sold for each country.
- We will specify the Country column as the grouping column:

```
df.groupby('Country').agg({'Quantity': 'sum'})
```

Performing Data Aggregation

Country	Quantity
Australia	83653
Austria	4827
Bahrain	260
Belgium	23152
Brazil	356
Canada	2763
Channel Islands	9479

Performing Data Aggregation

- We just need to provide the names of these columns as a list to the `.groupby()` method:
`df.groupby(['Country',
'StockCode']).agg({'Quantity': 'sum'})`

- You should get the following output:

		Quantity
Country	StockCode	
Australia	15036	600
	20665	6
	20675	216
	20676	216
	20677	216
...
Unspecified	85049A	1
	85179A	1
	85179C	1
	85180A	2
	85180B	1

19839 rows × 1 columns

Performing Data Aggregation

- To do so, we first need to create a new feature that will extract the date component of InvoiceDate (we just learned how to do this in the previous section):

```
df['Invoice_Date'] = df['InvoiceDate'].dt.date
```

- Then, we can add this new column in the .groupby() method:

```
df.groupby(['Country', 'StockCode', \
           'Invoice_Date']).agg({'Quantity': 'sum'})
```

Performing Data Aggregation

			Quantity
Country	StockCode	Invoice_Date	
Australia	15036	2011-05-17	600
	20665	2011-03-24	6
	20675	2011-01-06	72
		2011-03-03	144
	20676	2011-01-06	72
...
Unspecified	85049A	2011-07-28	1
	85179A	2011-07-28	1
	85179C	2011-07-28	1
	85180A	2011-07-28	2
	85180B	2011-07-28	1

310015 rows × 1 columns

Performing Data Aggregation

- To change it back to a single level, you need to call the `.reset_index()` method:

```
df_agg = df.groupby(['Country', 'StockCode', 'Invoice_Date'])\n            .agg({'Quantity': 'sum'}).reset_index()
```

```
df_agg.head()
```

- You should get the following output:

	Country	StockCode	Invoice_Date	Quantity
0	Australia	15036	2011-05-17	600
1	Australia	20665	2011-03-24	6
2	Australia	20675	2011-01-06	72
3	Australia	20675	2011-03-03	144
4	Australia	20676	2011-01-06	72

Performing Data Aggregation

- Now we can merge this new DataFrame into the original one using the `.merge()` method we saw earlier in this lesson:

```
df_merged = pd.merge(df, df_agg, how='left', \
                     on = ['Country', 'StockCode', \
                           'Invoice_Date'])
```

```
df_merged
```

Performing Data Aggregation

InvoiceNo	StockCode	Description	Quantity_x	InvoiceDate	UnitPrice	CustomerID	Country	Invoice_Date	Quantity_y
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	454
536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	33
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01	40
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	59

Performing Data Aggregation

- Let's replace the column names after merging with Quantity and DailyQuantity:

```
df_merged.rename(columns={"Quantity_x": "Quantity", \  
                     "Quantity_y": "DailyQuantity"}, \  
                     inplace=True)
```

```
df_merged
```

Performing Data Aggregation

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Invoice_Date	DailyQuantity
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	454
536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	33
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01	40

KNITTED

Performing Data Aggregation

- Now we can create a new feature that will calculate the ratio between the items sold with the daily total quantity of sold items in the corresponding country:

```
df_merged['QuantityRatio'] = df_merged['Quantity'] \  
    / df_merged['DailyQuantity']
```

`df_merged`

Performing Data Aggregation

- You should get this output:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Invoice_Date	DailyQuantity	QuantityRatio
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	454	0.013216
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	33	0.181818
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01	40	0.200000
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	59	0.101695
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	551	0.010889
...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France	2011-12-09	12	1.000000
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France	2011-12-09	6	1.000000
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-09	4	1.000000
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-09	4	1.000000
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France	2011-12-09	3	1.000000

541909 rows × 11 columns

Complete Exercise 12.04: Feature Engineering Using Data Aggregation on the AMES Housing Dataset

Complete Activity 12.01: Feature Engineering on a Financial Dataset

Summary

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

13. Imbalanced Datasets



Overview

- In this lesson, you will be dealing with imbalanced datasets, which are very prevalent in real-life scenarios.
- You will be using techniques such as SMOTE, MSMOTE, and random undersampling to address imbalanced datasets.

Introduction

Some of the use cases where we encounter imbalanced datasets include the following:

- Fraud detection for credit cards or insurance claims
- Medical diagnoses where we must detect the presence of rare diseases
- Intrusion detection in networks

Understanding the Business Context

- The business head of the bank for which you are working as a data scientist recently raised the alarm about the results of the term deposit propensity model that you built in lesson 3, Binary Classification.
- It has been observed that a large proportion of customers who were identified as potential cases for targeted marketing for term deposits have turned down the offer.

Complete Exercise 13.01: Benchmarking the Logistic Regression Model on the Dataset

Analysis of the Result

- To analyze the results obtained in the previous section, let's expand the confusion matrix in the form:

Actual	Predicted	
	Propensity: 'No'	Propensity: 'Yes'
Propensity: 'No'	True positive (TP) = 11707	False negative (FN) = 291
Propensity: 'Yes'	False positive (FP) = 1060	True negative (TN) = 506

Analysis of the Result

- The accuracy of the model is given by:

$$Accuracy \text{ of a model} = \frac{(TP + TN)}{(TP + FP + FN + TN)}$$

Analysis of the Result

- The precision value of any class is given by:

$$\text{Precision value} = \frac{\text{Correct prediction of the class}}{\text{Total predictions for that class}}$$

Analysis of the Result

- Similarly, the recall value for any class can be represented as follows:

$$\text{Recall value} = \frac{\text{Correct prediction of the class}}{\text{Total examples of the class}}$$

Analysis of the Result

- The following code will generate the percentages of the classes in the training data:

```
print('Percentage of negative class :',\n      (y_train[y_train=='yes'].value_counts()\n       /len(y_train) ) * 100)\nprint('Percentage of positive class :',\n      (y_train[y_train=='no'].value_counts()\n       /len(y_train) ) * 100)
```

Analysis of the Result

- You should get the following output:

Percentage of negative class: yes 11.764148

Name: y, dtype: float64

Percentage of positive class: no 88.235852

Name: y, dtype: float64

Challenges of Imbalanced Datasets

- Let's take, for example, a dataset where the negative class is around 99% and the positive class is 1% (as in a use case where a rare disease has to be detected, for instance).
- Have a look at the following code snippet:

Data set Size: 10,000 examples

Negative class : 9910

Positive Class : 90

Challenges of Imbalanced Datasets

- Suppose we had a poor classifier that was capable of only predicting the negative class; we would get the following confusion matrix:

Actual	Predicted	
	Propensity : 'Yes'	Propensity : 'No'
Probability of disease : 'Yes'	True positive (TP)= 0	False negative (FN) = 90
Probability of disease : 'No'	False positive (FP) = 0	True negative (TN) = 9900

Challenges of Imbalanced Datasets

- From the confusion matrix, let's calculate the accuracy measures.
- Have a look at the following code snippet:

```
# Classifier biased to only negative class  
Accuracy = (TP + TN) / ( TP + FP + FN + TN)  
= (0 + 9900) / ( 0 + 0 + 90 + 9900) = 9900/10000  
= 99%
```

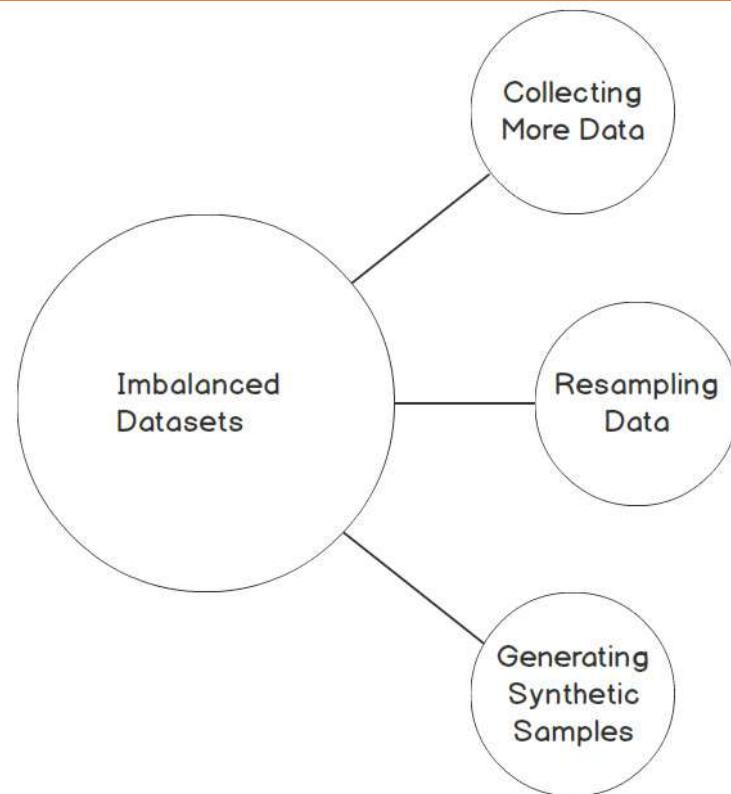
Challenges of Imbalanced Datasets

- Therefore, it is important to identify cases with imbalanced datasets and equally important to pick the right metric for analyzing such datasets.
- The right metric in this example would have been to look at the recall values for both the classes:

$$\begin{aligned}\text{Recall Positive class} &= \text{TP} / (\text{TP} + \text{FN}) = 0 / (0 + 90) \\ &= 0\end{aligned}$$

$$\begin{aligned}\text{Recall Negative Class} &= \text{TN} / (\text{TN} + \text{FP}) = 9900 / (9900 + 0) \\ &= 100\%\end{aligned}$$

Strategies for Dealing with Imbalanced Datasets



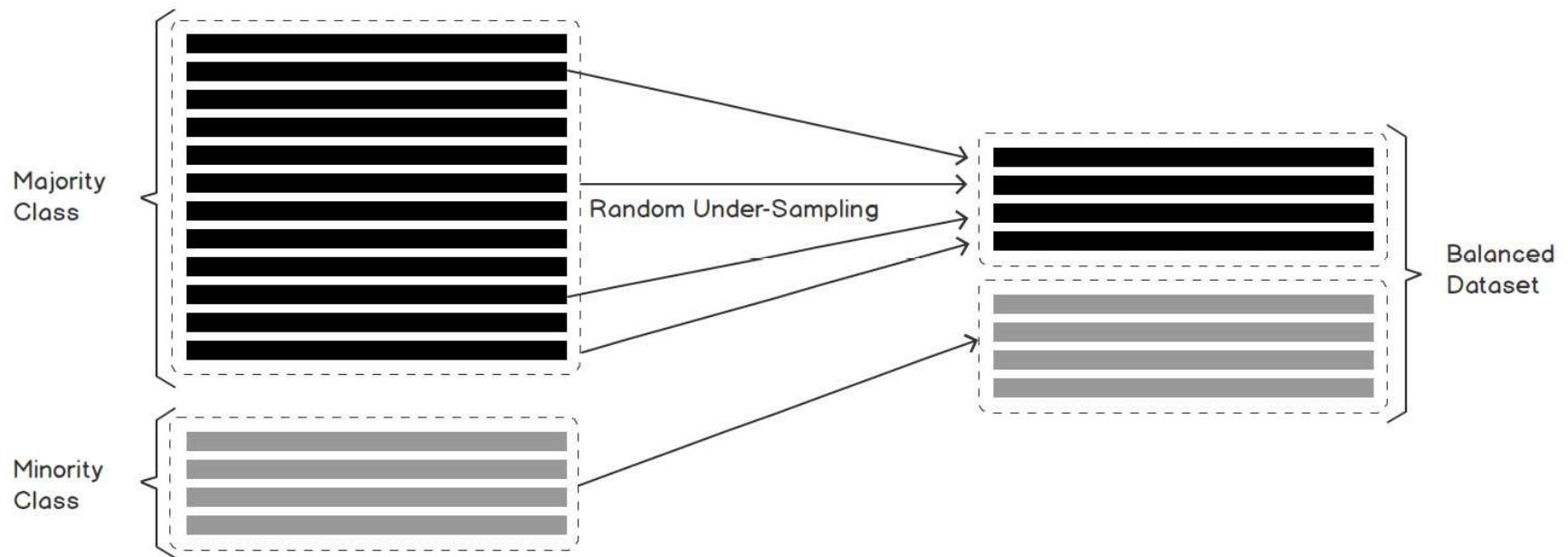
Collecting More Data

- Having encountered an imbalanced dataset, one of the first questions you need to ask is whether it is possible to get more data.
- This might appear naïve, but collecting more data, especially from the minority class, and then balancing the dataset should be the first strategy for addressing the class imbalance.

Resampling Data

- In many circumstances, collecting more data, especially from minority classes, can be challenging as data points for the minority class will be very minimal.
- In such circumstances, we need to adopt different strategies to work with our constraints and still strive to balance our dataset.
- One effective strategy is to resample our dataset to make the dataset more balanced.

Resampling Data



Complete Exercise 13.02: Implementing Random Undersampling and Classification on Our Banking Dataset to Find the Optimal Result

Analysis

- Let's analyze the results and compare them with those of the benchmark logistic regression model that we built at the beginning of this lesson.
- In the benchmark model, we had the problem of the model being biased toward the majority class with a very low recall value for the yes cases.

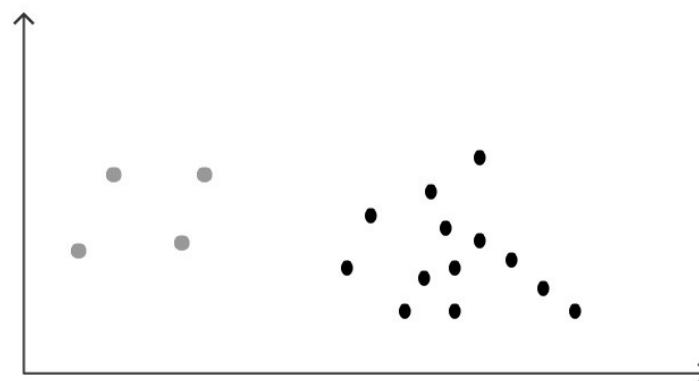
Generating Synthetic Samples

Two very popular methods for generating such synthetic points are:

- Synthetic Minority Oversampling Technique (SMOTE)
- Modified SMOTE (MSMOTE)

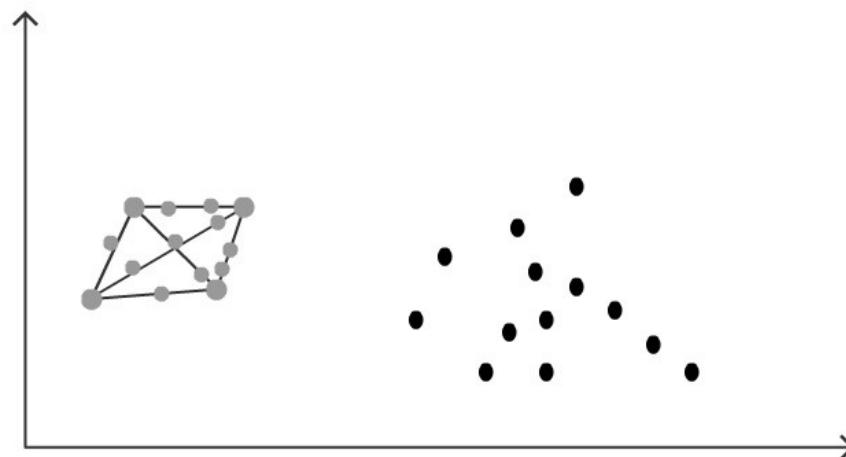
Generating Synthetic Samples

- The way the SMOTE algorithm generates synthetic data is by looking at the neighborhood of minority classes and generating new data points within the neighborhood:



Generating Synthetic Samples

- In creating synthetic points, an imaginary line connecting all the minority samples in the neighborhood is created and new data points are generated on this line, as shown, thereby balancing the dataset:



Implementation of SMOTE and MSMOTE

- SMOTE and MSMOTE can be implemented from a package called smote-variants in Python.
- The library can be installed through pip install in the Colab notecourse as shown here:

```
!pip install smote-variants
```

Complete Exercise 13.03: Implementing SMOTE on Our Banking Dataset to Find the Optimal Result

Complete Exercise 13.04: Implementing MSMOTE on Our Banking Dataset to Find the Optimal Result

Resampling Data

- Applying Balancing Techniques on a Telecom Dataset.
- Now that we have seen different balancing techniques, let's apply these techniques to a new dataset that is related to the churn of telecom customers.

Complete Activity 13.01: Finding the Best Balancing Technique by Fitting a Classifier on the Telecom Churn Dataset

Summary

- In this lesson, we learned about imbalanced datasets and strategies for addressing imbalanced datasets.
- We introduced the use cases where imbalanced datasets would be encountered.
- We looked at the challenges posed by imbalanced datasets and we were introduced to the metrics that should be used in the case of an imbalanced dataset.



14. Dimensionality Reduction



Overview

- This lesson introduces dimensionality reduction in data science.
- You will be using the Internet Advertisements dataset to analyze and evaluate different techniques in dimensionality reduction.
- By the end of this lesson, you will be able to analyze datasets with high dimensions and deal with the challenges posed by these datasets.

Introduction

This is the reality in some modern-day datasets in domains such as:

- Healthcare, where genetics datasets can have millions of features
- High-resolution imaging datasets
- Web data related to advertisements, ranking, and crawling

Introduction

- Storage and computation challenges: Large datasets with high dimensions require a lot of storage and expensive computational resources for analysis.
- Exploration challenges: When trying to explore data and derive insights, high-dimensional data can be really cumbersome.
- Algorithm challenges: Many algorithms do not scale well in high-dimensional settings.

Introduction

In this lesson, we will examine some of the popular dimensionality reduction techniques:

- Backward feature elimination or recursive feature elimination
- Forward feature selection
- Principal Component Analysis (PCA)
- Independent Component Analysis (ICA)
- Factor analysis

Business Context

- The marketing head of your company comes to you with a problem she has been grappling with.
- Many customers have been complaining about the browsing experience of your company's website because of the number of advertisements that pop up during browsing.
- Your company wants to build an engine on your web server that identifies potential advertisements and then eliminates them even before they pop up.

Complete Exercise 14.01: Loading and Cleaning the Dataset

Creating a High-Dimensional Dataset

- We will also calculate the time it takes for any activity using the `time()` function.
- Let's look at both these functions in action with a toy example.
- You begin by importing the necessary library functions:

```
import pandas as pd  
import numpy as np
```

Creating a High-Dimensional Dataset

- Then, to create a dummy data frame, we will use a small dataset with two rows and three columns for this example.
- We use the pd.np.array() function to create a data frame:

```
# Creating a simple data frame  
df = pd.np.array([[1, 2, 3], [4, 5, 6]])  
print(df.shape)  
df
```

- You should get the following output:

```
(2, 3)  
array([[1, 2, 3],  
       [4, 5, 6]])
```

Creating a High-Dimensional Dataset

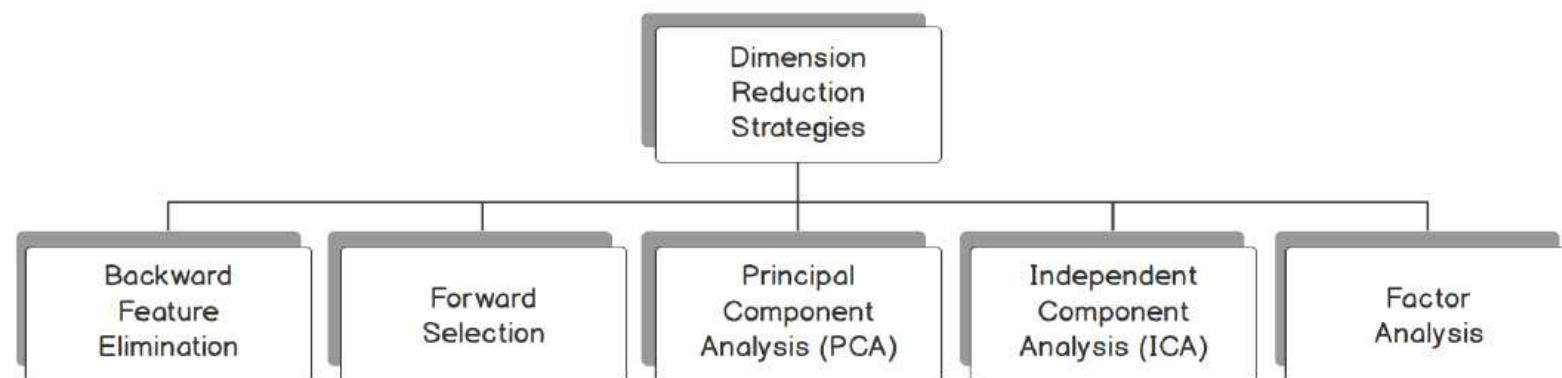
- Next, you replicate the dummy data frame and this replication of the columns is done using the pd.np.tile() function in the following code snippet:

```
# Replicating the data frame and noting the time
import time
# Starting a timing function
t0=time.time()
Newdf = pd.DataFrame(pd.np.tile(df, (1, 5)))
print(Newdf.shape)
print(Newdf)
# Finding the end time
print("Total time:", round(time.time()-t0, 3), "s")
```

Complete Activity 14.01: Fitting a Logistic Regression Model on a HighDimensional Dataset

Strategies for Addressing High-Dimensional Datasets

- Let's look at some of the techniques for dealing with high-dimensional datasets.
- In Figure, you can see the strategies we will be coming across in this lesson to deal with such datasets:



Backward Feature Elimination

- The mechanism behind the backward feature elimination algorithm is the recursive elimination of features and building a model on those features that remain after all the elimination.
- Initially, at a given iteration, the selected classification algorithm is first trained on all the n features available.
- For example, let's take the case of the original dataset we had, which had 1,558 features.
- The algorithm starts off with all the 1,558 features in the first iteration.

Complete Exercise 14.02: Dimensionality Reduction Using Backward Feature Elimination

Forward Feature Selection

In this process, we start off with an initial feature, and features are added one by one until no improvement in performance is achieved. The detailed process is as follows:

- Start model building with one feature.
- Iterate the model building process n times, each time selecting one feature at a time.
- The feature that gives the highest improvement in performance is selected.

Complete Exercise 14.03: Dimensionality Reduction Using Forward Feature Selection

Principal Component Analysis (PCA)

- PCA is a very effective dimensionality reduction technique that achieves dimensionality reduction without compromising on the information content of the data.
- The basic idea behind PCA is to first identify correlations among different variables within the dataset.
- Once correlations are identified, the algorithm decides to eliminate the variables in such a way that the variability of the data is maintained.
- In other words, PCA aims to find uncorrelated sources of data.

Principal Component Analysis (PCA)

- We will create a sample dataset with 2 variables and 100 random data points in each variable.
- Random data points are created using the rand() function.
- This is implemented in the following code:

```
import numpy as np
# Setting the seed for reproducibility
seed = np.random.RandomState(123)
# Generating an array of random numbers
X = seed.rand(100,2)
# Printing the shape of the dataset
X.shape
```

Principal Component Analysis (PCA)

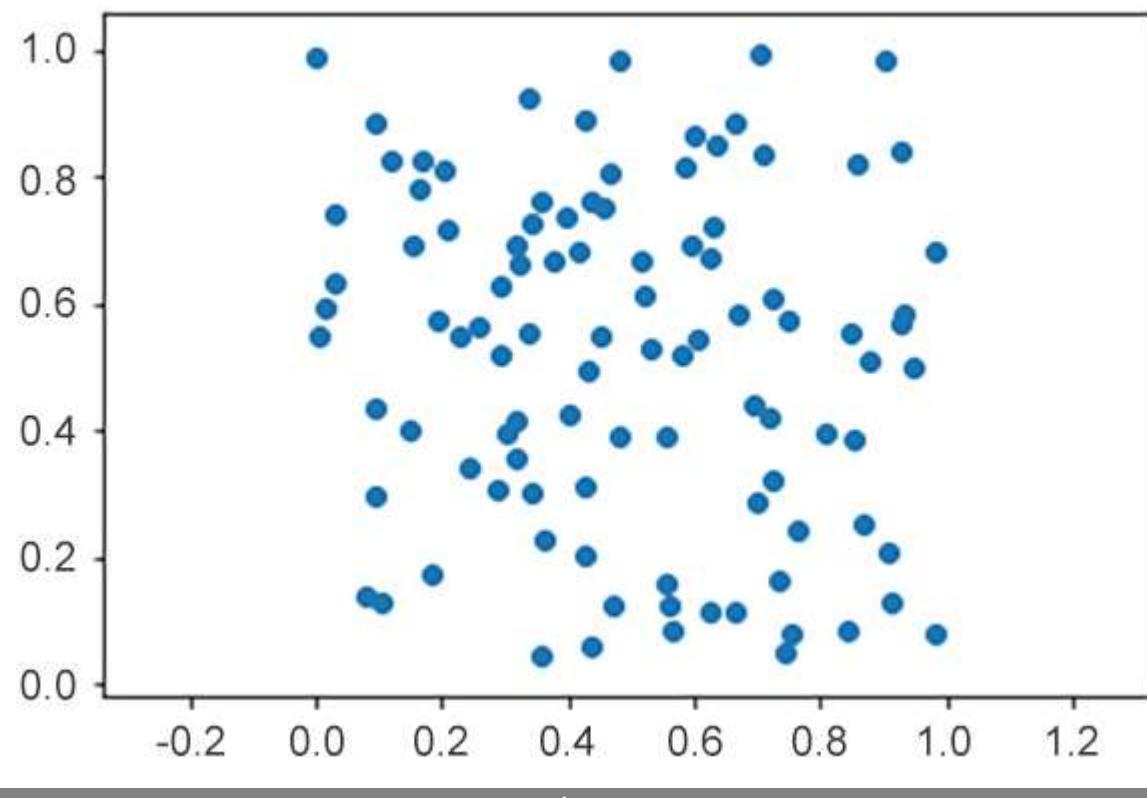
- Let's visualize this data using matplotlib:

```
import matplotlib.pyplot as plt  
%matplotlib inline  
plt.scatter(X[:, 0], X[:, 1])  
plt.axis('equal')
```

- You should get the following output:

```
(-0.04635361265714105,  
 1.0325632864350174,  
 -0.003996887112708292,  
 1.0429468329457663)
```

Principal Component Analysis (PCA)



Principal Component Analysis (PCA)

- This is implemented in code as follows:

```
from sklearn.decomposition import PCA  
# Defining one component  
pca = PCA(n_components=1)  
# Fitting the PCA function  
pca.fit(X)  
# Getting the new dataset  
X_pca = pca.transform(X)  
# Printing the shapes  
print("Original data set: ", X.shape)  
print("Data set after transformation:", X_pca.shape)
```

Principal Component Analysis (PCA)

- You should get the following output:

original shape: (100, 2)

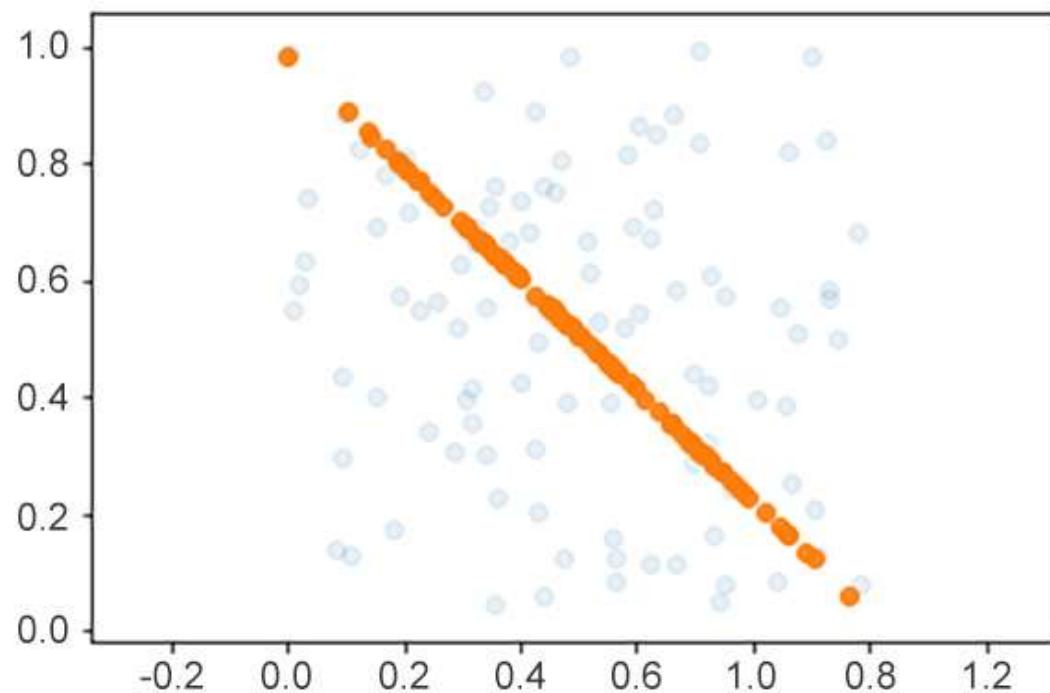
transformed shape: (100, 1)

Principal Component Analysis (PCA)

- To reverse the transformation, we use the `.inverse_transform()` function:

```
# Reversing the transformation and plotting
X_reverse = pca.inverse_transform(X_pca)
# Plotting the original data
plt.scatter(X[:, 0], X[:, 1], alpha=0.1)
# Plotting the reversed data
plt.scatter(X_reverse[:, 0], X_reverse[:, 1], alpha=0.9)
plt.axis('equal');
```

Principal Component Analysis (PCA)



Complete Exercise 14.04: Dimensionality Reduction Using PCA

Independent Component Analysis (ICA)

- ICA is a technique of dimensionality reduction that conceptually follows a similar path as PCA.
- Both ICA and PCA try to derive new sources of data by linearly combining the original data.
- However, the difference between them lies in the method they use to find new sources of data.
- While PCA attempts to find uncorrelated sources of data, ICA attempts to find independent sources of data.

Complete Exercise 14.05: Dimensionality Reduction Using Independent Component Analysis

Factor Analysis

- Factor analysis is a technique that achieves dimensionality reduction by grouping variables that are highly correlated.
- Let's look at an example from our context of predicting advertisements.
- In our dataset, there could be many features that describe the geometry (the size and shape of an image in the ad) of the images on a web page.

Complete Exercise 14.06: Dimensionality Reduction Using Factor Analysis

Comparing Different Dimensionality Reduction Techniques

- We will randomly sample some data points from a known distribution and then add these random samples to the existing dataset to create a new dataset.
- Let's carry out an experiment to see how a new dataset can be created from an existing dataset.
- We import the necessary libraries:

```
import pandas as pd  
import numpy as np
```

Comparing Different Dimensionality Reduction Techniques

- We will use a small dataset with two rows and three columns for this example.
- We use the pd.np.array() function to create a data frame:

```
# Creating a simple data frame  
df = pd.np.array([[1, 2, 3], [4, 5, 6]])  
print(df.shape)  
df
```

- You should get the following output:

```
(2, 3)  
array([[1, 2, 3],  
       [4, 5, 6]])
```

Comparing Different Dimensionality Reduction Techniques

- Let's see how this is implemented in code:

```
# Defining the mean and standard deviation
```

```
mu, sigma = 0, 0.1
```

```
# Generating random sample
```

```
noise = np.random.normal(mu, sigma, [2,3])
```

```
noise.shape
```

- You should get the following output:

```
(2, 3)
```

Comparing Different Dimensionality Reduction Techniques

- Print the sampled data frame:

```
# Sampled data frame  
noise
```

- You will get something like the following output:

```
array([[-0.07175021, -0.21135372,  0.10258917],  
      [ 0.03737542,  0.00045449, -0.04866098]])
```

Comparing Different Dimensionality Reduction Techniques

- The next step is to add the original data frame and the sampled data frame to get the new dataset:

```
# Creating a new data set by adding sampled data  
frame
```

```
df_new = df + noise
```

```
df_new
```

- You should get something like the following output:

```
array([[0.92824979, 1.78864628, 3.10258917],  
       [4.03737542, 5.00045449, 5.95133902]])
```

Complete Activity 14.02: Comparison of Dimensionality Reduction Techniques on the Enhanced Ads Dataset

Summary

- In this lesson, we have learned about various techniques for dimensionality reduction.
- Let's summarize what we have learned in this lesson.
- At the beginning of the lesson, we were introduced to the challenges inherent with some of the modern-day datasets in terms of scalability.

15. Ensemble Learning



Overview

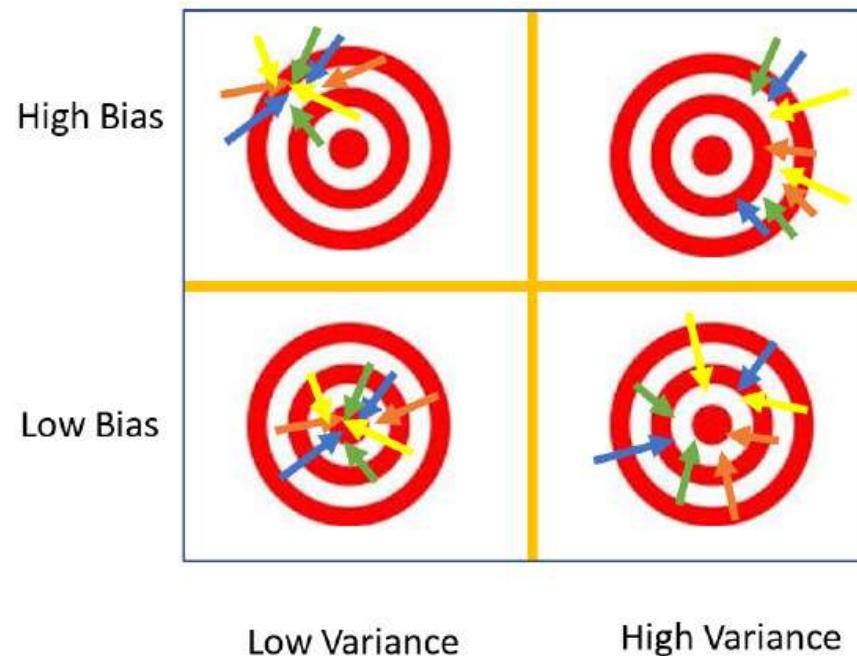
- By the end of this lesson, you will be able to describe ensemble learning and apply different ensemble learning techniques to your dataset.
- You will also be able to fit a dataset on a model and analyze the results after ensemble learning.

Introduction

In this lesson, we will further enhance our repertoire of skills with another set of techniques, called ensemble learning, in which we will be dealing with different ensemble learning techniques such as the following:

- Averaging
- Weighted averaging
- Max voting
- Bagging
- Boosting
- Blending

Ensemble Learning



Variance

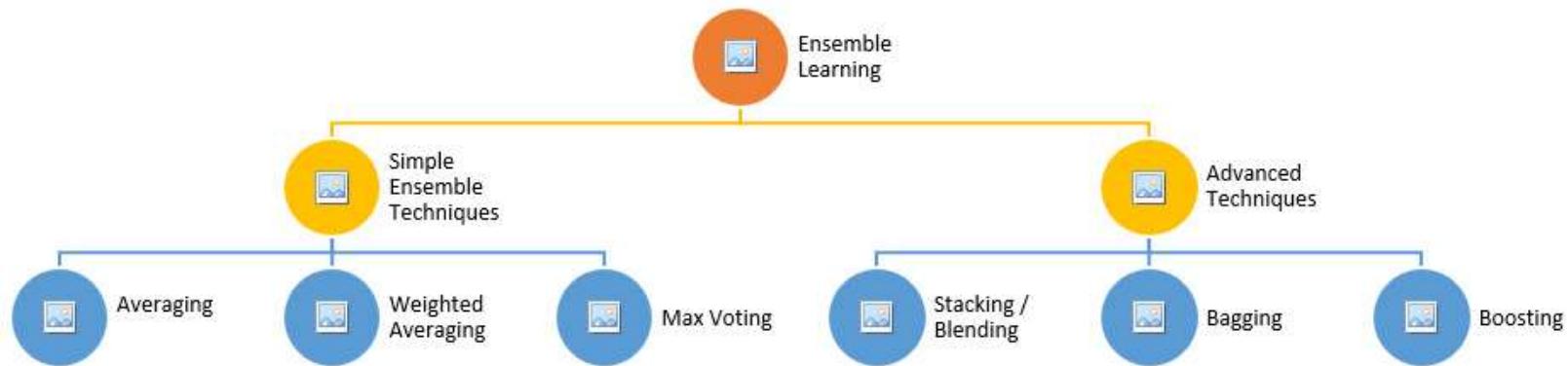
- Variance is the measure of how spread out data is.
- In the context of machine learning, models with high variance imply that the predictions generated on the same test set will differ considerably when different training sets are used to fit the model.
- The underlying reason for high variability could be attributed to the model being attuned to specific nuances of training data rather than generalizing the relationship between input and output.

Bias

- Bias is the difference between the ground truth and the average value of our predictions.
- A low bias will indicate that the predictions are very close to the actual values.
- A high bias implies that the model has oversimplified the relationship between the inputs and outputs, leading to high error rates on test sets, which again is an undesirable outcome.

Bias

- In this lesson, we will learn about different ensemble techniques, which can be classified into two types, that is, simple and advanced techniques:



Business Context

- You are working in the credit card division of your bank.
- The operations head of your company has requested your help in determining whether a customer is creditworthy or not.
- You have been provided with credit card operations data.

Complete Exercise 15.01: Loading, Exploring, and Cleaning the Data

Complete Activity 15.01: Fitting a Logistic Regression Model on Credit Card Data

Simple Methods for Ensemble Learning

As defined earlier in the lesson, ensemble learning is all about combining the strengths of individual models to get a superior model.

In this section, we will explore some simple techniques such as the following:

- Averaging
- Weighted averaging
- Max voting

Averaging

- If we were to output the probability of each class for our benchmark model, we would get two outputs for each example corresponding to the probability for each class.
- This is demonstrated in an example prediction in the following table:

Example	Class: '1'	Class: '0'
Test set example 1	0.902	0.098
Test set example 2	0.401	0.559
Test set example 3	0.732	0.268

Complete Exercise 15.02: Ensemble Model Using the Averaging Technique

Weighted Averaging

- Weighted averaging is an extension of the averaging method that we saw earlier.
- The major difference in both of these approaches is in the way the combined predictions are generated.
- In the weighted averaging method, we assign weights to each model's predictions and then generate the combined predictions.
- The weights are assigned based on our judgment of which model would be the most influential in the ensemble

Complete Exercise 15.03: Ensemble Model Using the Weighted Averaging Technique

Max Voting

- Let's say we have three individual learners who learned on the training set.
- Each of them generates their predictions on the test set, which is tabulated in the following table.
- The predictions are either for class '1' or class '0':

Test Example	Prediction of learner 1	Prediction of learner 2	Prediction of learner 3	Final Prediction
Example 1	1	1	1	1
Example 2	1	0	0	0
Example 3	1	1	0	1
Example 4	0	1	0	0

Complete Exercise 15.04: Ensemble Model Using Max Voting

Advanced Techniques for Ensemble Learning

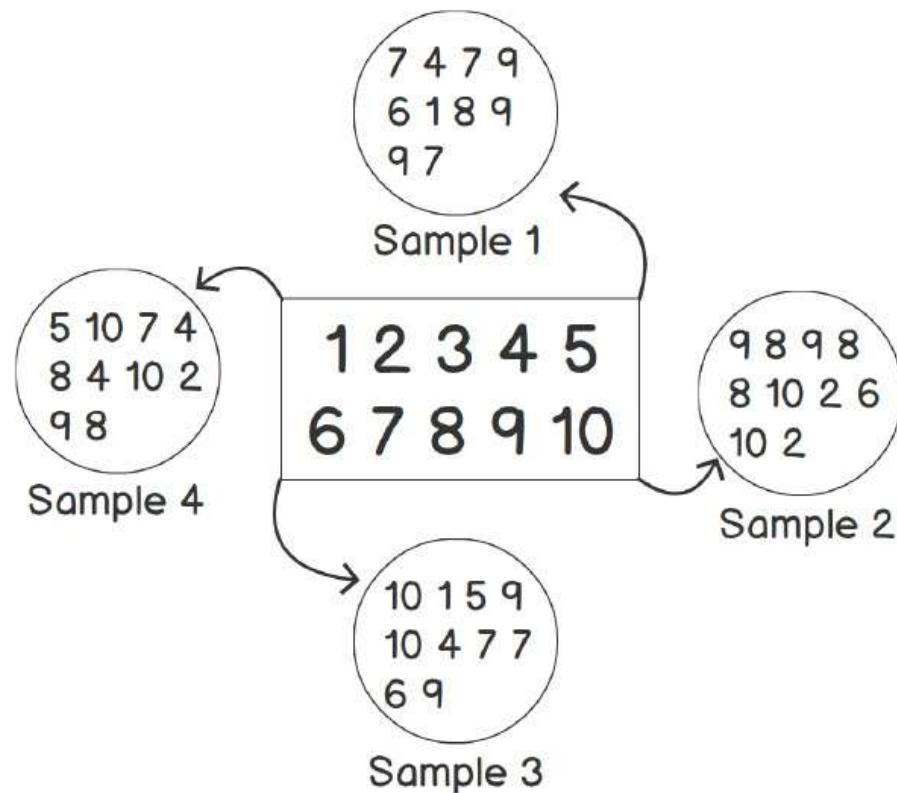
Having learned simple techniques for ensemble learning, let's now explore some advanced techniques. Among the advanced techniques, we will be dealing with three different kinds of ensemble learning:

- Bagging
- Boosting
- Stacking/blending

Bagging

- Bagging is a pseudonym for Bootstrap Aggregating. Before we explain how bagging works, let's describe what bootstrapping is.
- Bootstrapping has its etymological origins in the phrase, Pulling oneself up by one's bootstrap.
- The essence of this phrase is to make the best use of the available resources.

Bagging



Complete Exercise 15.05: Ensemble Learning Using Bagging

Boosting

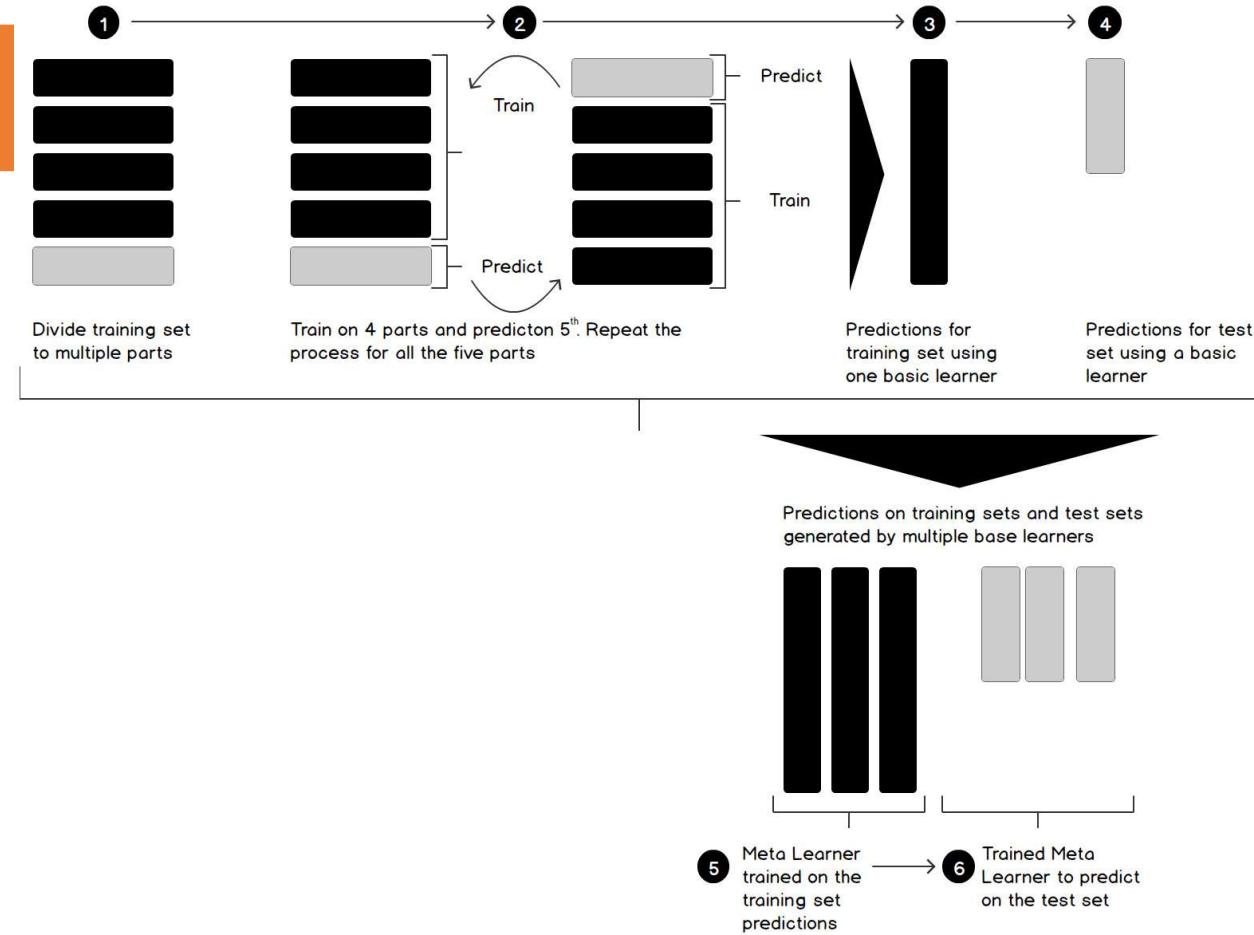
- The bagging technique, which we discussed in the last section, can be termed as a parallel learning technique.
- This means that each base learner is fit independently of the other and their predictions are aggregated.
- Unlike the bagging method, boosting works in a sequential manner.
- It works on the principle of correcting the prediction errors of each base learner.

Complete Exercise 15.06: Ensemble Learning Using Boosting

Stacking

- Stacking, in principle, works in a similar way to bagging and boosting in that it combines base learners to form a meta learner.
- However, the approach for getting the meta learners from the base learners differs substantially in stacking.
- In stacking, the meta learner is fit on the predictions made by the base learners

Stacking



Complete Exercise 15.07: Ensemble Learning Using Stacking

Complete Activity 15.02: Comparison of Advanced Ensemble Techniques

Summary

- In this lesson, we learned about various techniques of ensemble learning.
- Let's summarize our learning in this lesson.
- At the beginning of the lesson, we were introduced to the concepts of variance and bias and we learned that ensemble learning is a technique that aims to combine individual models to create a superior model, thereby reducing variance and bias and improving performance.

THANK YOU !!