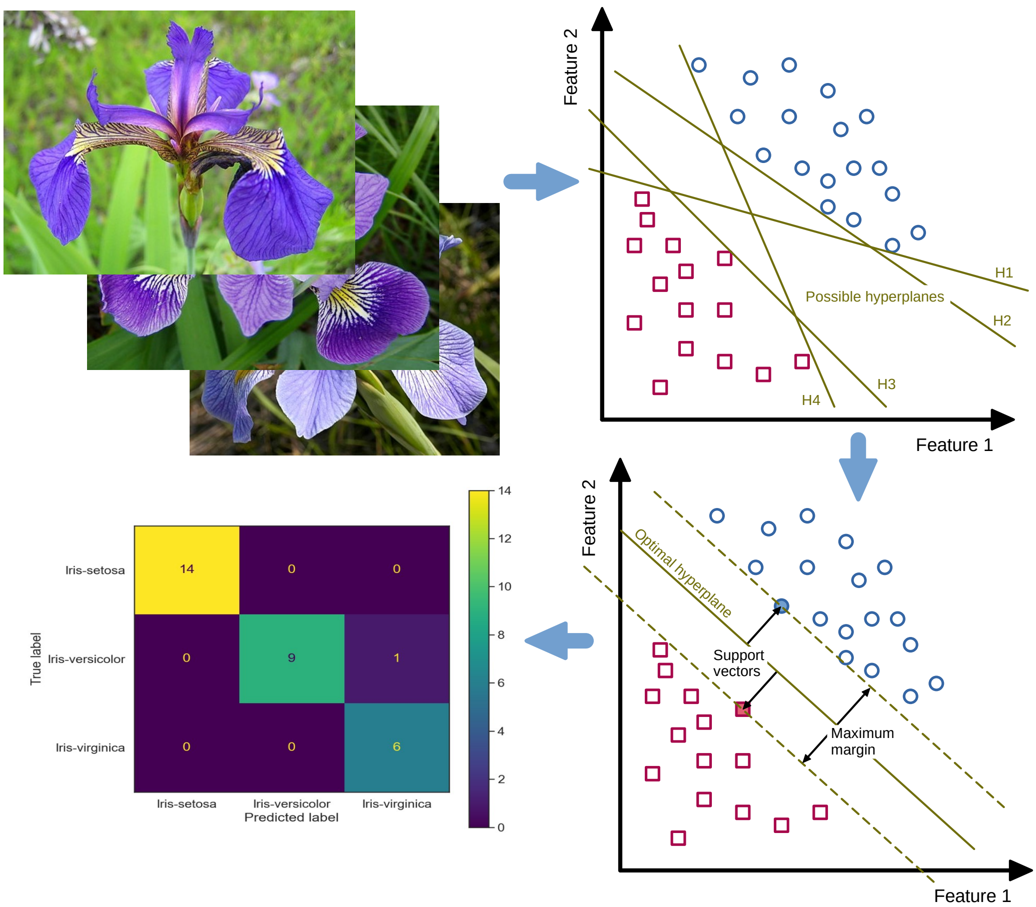
# Getting started with Support Vector Classifiers (SVC) - A systematic step-by-step approach

Dipl.-Ing. Björn Kasper (kasper.bjoern@bgetem.de)

*Test and Certification Body for Electrical Engineering at BG ETEM*

August 4, 2022




Anyone who wants to seriously deal with the hypothetical topic of our time "Artificial Intelligence (AI)" or "Machine Learning (ML)" cannot avoid dealing with the basic ML algorithms, corresponding software tools, libraries and programming systems. However, someone who opens the door for the first time to this equally very exciting and arbitrarily complex and, at first glance, confusing world will very quickly be overwhelmed. Here, it is a good idea to consult introductory and systematic tutorials. Therefore, this Getting Started tutorial systematically demonstrates the typical ML work process step-by-step using the very powerful and performant "Support Vector Classifier (SVC)" and the widely known and very beginner-friendly "Iris Dataset". Furthermore, the selection of the "correct" SVC kernel and its parameters are described and their effect on the classification result is shown.

# Contents



# 1 Introduction

Anyone who wants to seriously deal with the hypothetical topic of our time **Artificial Intelligence (AI)** or **Machine Learning (ML)** cannot avoid dealing with the basic ML algorithms, corresponding software tools, libraries and programming systems.

However, someone who opens the door for the first time to this equally very exciting and arbitrarily complex and, at first glance, confusing world will very quickly be overwhelmed. Here, it is a good idea to consult introductory and systematic tutorials.

Therefore, this Getting Started tutorial systematically demonstrates the typical ML work process step-by-step using the very powerful and performant **Support Vector Classifier (SVC)** and the widely known and very beginner-friendly **Iris Dataset**.

This tutorial will be presented as part of a workshop at the DGUV symposium **Artificial Intelligence**, probably in November 2022 in Dresden. The workshop addresses interested ML novices.

For the target audience in the workshop, the SVC algorithm was intentionally chosen to show that there are many other very powerful and performant ML algorithms apart from the **deep neural networks** that are very present in the media. On the other hand, a necessary and comprehensible introduction to neural networks for newcomers would not be possible within the time frame given for the workshop.

Furthermore, this tutorial does *not* address the generation or acquisition of ML-ready datasets. A newcomer to ML will (or should) first try to familiarize himself with ML algorithms, tools, libraries and programming systems. Only then it makes sense to explore one's own environment with respect to ML-suitable applications and to acquire suitable data sets from them.

Therefore, the tutorial will demonstrate the usage of selected ML tools in the form of Python libraries as well as the systematic approach to the widely known and very beginner-friendly **Iris dataset**.

Furthermore, the selection of the "correct" SVC kernel and its parameters are described and their effect on the classification result is shown.

The following steps of the systematic ML process are covered in the next main sections:

- STEP 0: Get the data
- STEP 1: Exploring the data
- STEP 2: Prepare the data
- STEP 3: Classify by support vector classifier - SVC
- STEP 4: Evaluate the results - metrics
- STEP 5: Vary parameters

## 2 Load globally used libraries and set plot parameters

[1]:
```
import time

from IPython.display import HTML

import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm, metrics
import seaborn as sns
%matplotlib inline
```

## 3 STEP 0: Get the data

Since this is intended to be an introduction to the world of machine learning (ML), this step does NOT deal with the design of an application suitable for ML and the acquisition of valid measurement data.

In order to get to know the typical work steps and ML tools, the use of **well-known and well-researched data sets** is clearly **recommended**.

In the further course, the famous Iris flower data sets will be used. It can be downloaded on Iris Flower Dataset | Kaggle. Furthermore, the dataset is included in Python in the machine learning package Scikit-learn, so that users can access it without having to find a special source for it.

[2]:
```
# import some data to play with
irisdata_df = pd.read_csv('./datasets/IRIS_flower_dataset_kaggle.csv')
```

# 4 STEP 1: Exploring the data

## 4.1 Goals of exploration

The objectives of the exploration of the dataset are as follows:

1. Clarify the **origins history**:
   - Where did the data come from? => Contact persons and licensing permissions?
   - Who obtained the data and with which (measurement) methods? => Did systematic errors occur during the acquisition?
   - What were they originally intended for? => Can they be used for my application?
2. Overview of the internal **structure and organisation** of the data:
   - Which columns are there? => With which methods can they be read in (e.g. import of CSV files)?
   - What do they contain for (physical) measured variables? => Which technical or physical correlations exist?
   - Which data formats or types are there? => Do they have to be converted?
   - In which value ranges do the measurement data vary? => Are normalizations necessary?
3. Identify **anomalies** in the data sets:
   - Do the data have **gaps** or **duplicates**? => Does the data set needs to be cleaned?
   - Are there obvious erroneous entries or measurement outliers? => Does (statistical) filtering have to be carried out?
4. Avoidance of **tendencies due to bias**:
   - Are all possible classes included in the dataset and equally distributed? => Does the data set need to be enriched with additional data for balance?
5. Find a first rough **idea of which correlations** could be in the data set

## 4.2 Clarify the origins history

> The ***Iris* flower data sets** is a multivariate data set introduced by the British statistician and biologist *Ronald Fisher* in his paper "The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis" (1936). It is sometimes called *Anderson's Iris data set* because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species (source: Iris flower data set).

The dataset is published in Public Domain with a CC0-License.

This dataset became a typical test case for many statistical classification techniques in machine learning such as **support vector machines**.

> [..] measurements of the flowers of fifty plants each of the two species *Iris setosa* and *I. versicolor*, found **growing together in the same colony** and measured by Dr E. Anderson [..] (source: R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". Annals of Eugenics)

> [..] *Iris virginica*, differs from the two other samples in **not being taken from the same natural colony** [..] (source: ibidem)

## 4.3 Overview of the internal structure and organisation of the data

The data set consists of 50 samples from each of three species of Iris (*Iris setosa*, *Iris virginica* and *Iris versicolor*), so there are 150 total samples. Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.
Here is a principle illustration of a flower with sepal and petal:

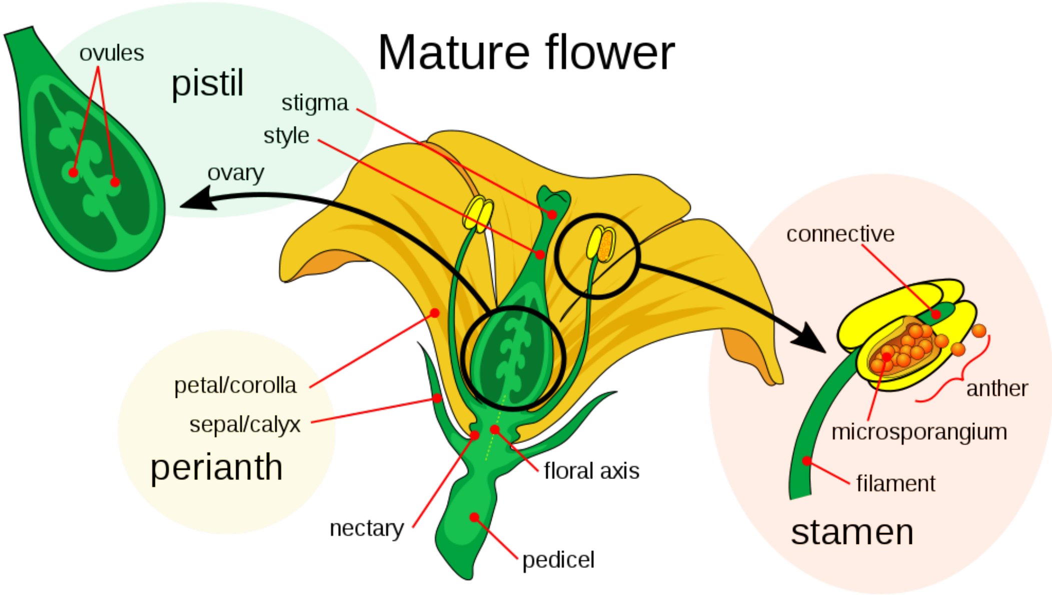



Figure 1: Principle illustration of a flower with sepal and petal (source: Mature_flower_diagram.svg, license: public domain)

Here are pictures of the three different Iris species (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Given the dimensions of the flower, it will be possible to predict the class of the flower.

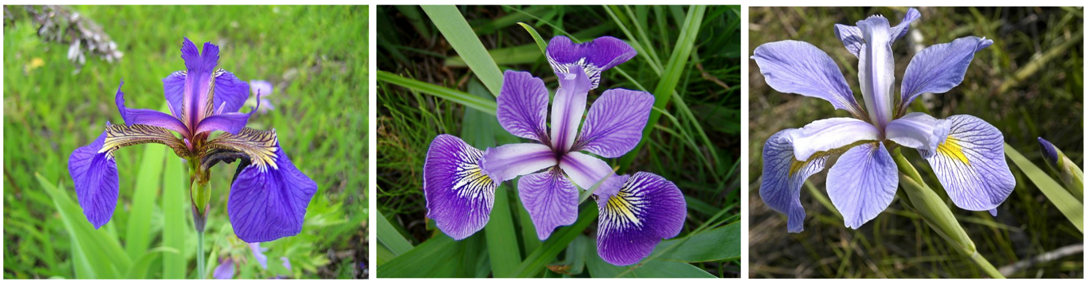


Figure 2: left: *Iris setosa* (source: Irissetosa1.jpg, license: public domain); middle: *Iris versicolor* (source: Iris_versicolor_3.jpg, license: CC-SA 3.0); right: *Iris virginica* (source: Iris_virginica.jpg, license: CC-SA 2.0)

#### 4.3.1 Inspect structure of dataframe

Print first or last 5 rows of dataframe:

[3]:
```
irisdata_df.head()
```

[3]:

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

[4]:
```
irisdata_df.tail()
```

[4]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |

```
146            6.3          2.5           5.0          1.9  Iris-virginica
147            6.5          3.0           5.2          2.0  Iris-virginica
148            6.2          3.4           5.4          2.3  Iris-virginica
149            5.9          3.0           5.1          1.8  Iris-virginica
```

While printing a dataframe - only an abbreviated view of the dataframe is shown :(
Default setting in the pandas library makes it to display only 5 lines from head and from tail.

[5]:
```
irisdata_df
```

[5]:
```
     sepal_length  sepal_width  petal_length  petal_width         species
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            …            …             …            …             …
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

To print all rows of a dataframe, the option `display.max_rows` has to set to `None` in pandas:

[6]:
```
pd.set_option('display.max_rows', None)
irisdata_df
```

[6]:
```
     sepal_length  sepal_width  petal_length  petal_width          species
0             5.1          3.5           1.4          0.2      Iris-setosa
1             4.9          3.0           1.4          0.2      Iris-setosa
2             4.7          3.2           1.3          0.2      Iris-setosa
3             4.6          3.1           1.5          0.2      Iris-setosa
4             5.0          3.6           1.4          0.2      Iris-setosa
5             5.4          3.9           1.7          0.4      Iris-setosa
6             4.6          3.4           1.4          0.3      Iris-setosa
7             5.0          3.4           1.5          0.2      Iris-setosa
8             4.4          2.9           1.4          0.2      Iris-setosa
9             4.9          3.1           1.5          0.1      Iris-setosa
10            5.4          3.7           1.5          0.2      Iris-setosa
11            4.8          3.4           1.6          0.2      Iris-setosa
12            4.8          3.0           1.4          0.1      Iris-setosa
13            4.3          3.0           1.1          0.1      Iris-setosa
14            5.8          4.0           1.2          0.2      Iris-setosa
15            5.7          4.4           1.5          0.4      Iris-setosa
16            5.4          3.9           1.3          0.4      Iris-setosa
17            5.1          3.5           1.4          0.3      Iris-setosa
18            5.7          3.8           1.7          0.3      Iris-setosa
19            5.1          3.8           1.5          0.3      Iris-setosa
20            5.4          3.4           1.7          0.2      Iris-setosa
21            5.1          3.7           1.5          0.4      Iris-setosa
22            4.6          3.6           1.0          0.2      Iris-setosa
23            5.1          3.3           1.7          0.5      Iris-setosa
24            4.8          3.4           1.9          0.2      Iris-setosa
25            5.0          3.0           1.6          0.2      Iris-setosa
26            5.0          3.4           1.6          0.4      Iris-setosa
```

| | | | | | |
|---|---|---|---|---|---|
| 27 | 5.2 | 3.5 | 1.5 | 0.2 | Iris-setosa |
| 28 | 5.2 | 3.4 | 1.4 | 0.2 | Iris-setosa |
| 29 | 4.7 | 3.2 | 1.6 | 0.2 | Iris-setosa |
| 30 | 4.8 | 3.1 | 1.6 | 0.2 | Iris-setosa |
| 31 | 5.4 | 3.4 | 1.5 | 0.4 | Iris-setosa |
| 32 | 5.2 | 4.1 | 1.5 | 0.1 | Iris-setosa |
| 33 | 5.5 | 4.2 | 1.4 | 0.2 | Iris-setosa |
| 34 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 35 | 5.0 | 3.2 | 1.2 | 0.2 | Iris-setosa |
| 36 | 5.5 | 3.5 | 1.3 | 0.2 | Iris-setosa |
| 37 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 38 | 4.4 | 3.0 | 1.3 | 0.2 | Iris-setosa |
| 39 | 5.1 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| 40 | 5.0 | 3.5 | 1.3 | 0.3 | Iris-setosa |
| 41 | 4.5 | 2.3 | 1.3 | 0.3 | Iris-setosa |
| 42 | 4.4 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 43 | 5.0 | 3.5 | 1.6 | 0.6 | Iris-setosa |
| 44 | 5.1 | 3.8 | 1.9 | 0.4 | Iris-setosa |
| 45 | 4.8 | 3.0 | 1.4 | 0.3 | Iris-setosa |
| 46 | 5.1 | 3.8 | 1.6 | 0.2 | Iris-setosa |
| 47 | 4.6 | 3.2 | 1.4 | 0.2 | Iris-setosa |
| 48 | 5.3 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| 49 | 5.0 | 3.3 | 1.4 | 0.2 | Iris-setosa |
| 50 | 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| 51 | 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| 52 | 6.9 | 3.1 | 4.9 | 1.5 | Iris-versicolor |
| 53 | 5.5 | 2.3 | 4.0 | 1.3 | Iris-versicolor |
| 54 | 6.5 | 2.8 | 4.6 | 1.5 | Iris-versicolor |
| 55 | 5.7 | 2.8 | 4.5 | 1.3 | Iris-versicolor |
| 56 | 6.3 | 3.3 | 4.7 | 1.6 | Iris-versicolor |
| 57 | 4.9 | 2.4 | 3.3 | 1.0 | Iris-versicolor |
| 58 | 6.6 | 2.9 | 4.6 | 1.3 | Iris-versicolor |
| 59 | 5.2 | 2.7 | 3.9 | 1.4 | Iris-versicolor |
| 60 | 5.0 | 2.0 | 3.5 | 1.0 | Iris-versicolor |
| 61 | 5.9 | 3.0 | 4.2 | 1.5 | Iris-versicolor |
| 62 | 6.0 | 2.2 | 4.0 | 1.0 | Iris-versicolor |
| 63 | 6.1 | 2.9 | 4.7 | 1.4 | Iris-versicolor |
| 64 | 5.6 | 2.9 | 3.6 | 1.3 | Iris-versicolor |
| 65 | 6.7 | 3.1 | 4.4 | 1.4 | Iris-versicolor |
| 66 | 5.6 | 3.0 | 4.5 | 1.5 | Iris-versicolor |
| 67 | 5.8 | 2.7 | 4.1 | 1.0 | Iris-versicolor |
| 68 | 6.2 | 2.2 | 4.5 | 1.5 | Iris-versicolor |
| 69 | 5.6 | 2.5 | 3.9 | 1.1 | Iris-versicolor |
| 70 | 5.9 | 3.2 | 4.8 | 1.8 | Iris-versicolor |
| 71 | 6.1 | 2.8 | 4.0 | 1.3 | Iris-versicolor |
| 72 | 6.3 | 2.5 | 4.9 | 1.5 | Iris-versicolor |
| 73 | 6.1 | 2.8 | 4.7 | 1.2 | Iris-versicolor |
| 74 | 6.4 | 2.9 | 4.3 | 1.3 | Iris-versicolor |
| 75 | 6.6 | 3.0 | 4.4 | 1.4 | Iris-versicolor |
| 76 | 6.8 | 2.8 | 4.8 | 1.4 | Iris-versicolor |
| 77 | 6.7 | 3.0 | 5.0 | 1.7 | Iris-versicolor |
| 78 | 6.0 | 2.9 | 4.5 | 1.5 | Iris-versicolor |
| 79 | 5.7 | 2.6 | 3.5 | 1.0 | Iris-versicolor |
| 80 | 5.5 | 2.4 | 3.8 | 1.1 | Iris-versicolor |
| 81 | 5.5 | 2.4 | 3.7 | 1.0 | Iris-versicolor |
| 82 | 5.8 | 2.7 | 3.9 | 1.2 | Iris-versicolor |
| 83 | 6.0 | 2.7 | 5.1 | 1.6 | Iris-versicolor |

| | | | | | |
|---|---|---|---|---|---|
| 84 | 5.4 | 3.0 | 4.5 | 1.5 | Iris-versicolor |
| 85 | 6.0 | 3.4 | 4.5 | 1.6 | Iris-versicolor |
| 86 | 6.7 | 3.1 | 4.7 | 1.5 | Iris-versicolor |
| 87 | 6.3 | 2.3 | 4.4 | 1.3 | Iris-versicolor |
| 88 | 5.6 | 3.0 | 4.1 | 1.3 | Iris-versicolor |
| 89 | 5.5 | 2.5 | 4.0 | 1.3 | Iris-versicolor |
| 90 | 5.5 | 2.6 | 4.4 | 1.2 | Iris-versicolor |
| 91 | 6.1 | 3.0 | 4.6 | 1.4 | Iris-versicolor |
| 92 | 5.8 | 2.6 | 4.0 | 1.2 | Iris-versicolor |
| 93 | 5.0 | 2.3 | 3.3 | 1.0 | Iris-versicolor |
| 94 | 5.6 | 2.7 | 4.2 | 1.3 | Iris-versicolor |
| 95 | 5.7 | 3.0 | 4.2 | 1.2 | Iris-versicolor |
| 96 | 5.7 | 2.9 | 4.2 | 1.3 | Iris-versicolor |
| 97 | 6.2 | 2.9 | 4.3 | 1.3 | Iris-versicolor |
| 98 | 5.1 | 2.5 | 3.0 | 1.1 | Iris-versicolor |
| 99 | 5.7 | 2.8 | 4.1 | 1.3 | Iris-versicolor |
| 100 | 6.3 | 3.3 | 6.0 | 2.5 | Iris-virginica |
| 101 | 5.8 | 2.7 | 5.1 | 1.9 | Iris-virginica |
| 102 | 7.1 | 3.0 | 5.9 | 2.1 | Iris-virginica |
| 103 | 6.3 | 2.9 | 5.6 | 1.8 | Iris-virginica |
| 104 | 6.5 | 3.0 | 5.8 | 2.2 | Iris-virginica |
| 105 | 7.6 | 3.0 | 6.6 | 2.1 | Iris-virginica |
| 106 | 4.9 | 2.5 | 4.5 | 1.7 | Iris-virginica |
| 107 | 7.3 | 2.9 | 6.3 | 1.8 | Iris-virginica |
| 108 | 6.7 | 2.5 | 5.8 | 1.8 | Iris-virginica |
| 109 | 7.2 | 3.6 | 6.1 | 2.5 | Iris-virginica |
| 110 | 6.5 | 3.2 | 5.1 | 2.0 | Iris-virginica |
| 111 | 6.4 | 2.7 | 5.3 | 1.9 | Iris-virginica |
| 112 | 6.8 | 3.0 | 5.5 | 2.1 | Iris-virginica |
| 113 | 5.7 | 2.5 | 5.0 | 2.0 | Iris-virginica |
| 114 | 5.8 | 2.8 | 5.1 | 2.4 | Iris-virginica |
| 115 | 6.4 | 3.2 | 5.3 | 2.3 | Iris-virginica |
| 116 | 6.5 | 3.0 | 5.5 | 1.8 | Iris-virginica |
| 117 | 7.7 | 3.8 | 6.7 | 2.2 | Iris-virginica |
| 118 | 7.7 | 2.6 | 6.9 | 2.3 | Iris-virginica |
| 119 | 6.0 | 2.2 | 5.0 | 1.5 | Iris-virginica |
| 120 | 6.9 | 3.2 | 5.7 | 2.3 | Iris-virginica |
| 121 | 5.6 | 2.8 | 4.9 | 2.0 | Iris-virginica |
| 122 | 7.7 | 2.8 | 6.7 | 2.0 | Iris-virginica |
| 123 | 6.3 | 2.7 | 4.9 | 1.8 | Iris-virginica |
| 124 | 6.7 | 3.3 | 5.7 | 2.1 | Iris-virginica |
| 125 | 7.2 | 3.2 | 6.0 | 1.8 | Iris-virginica |
| 126 | 6.2 | 2.8 | 4.8 | 1.8 | Iris-virginica |
| 127 | 6.1 | 3.0 | 4.9 | 1.8 | Iris-virginica |
| 128 | 6.4 | 2.8 | 5.6 | 2.1 | Iris-virginica |
| 129 | 7.2 | 3.0 | 5.8 | 1.6 | Iris-virginica |
| 130 | 7.4 | 2.8 | 6.1 | 1.9 | Iris-virginica |
| 131 | 7.9 | 3.8 | 6.4 | 2.0 | Iris-virginica |
| 132 | 6.4 | 2.8 | 5.6 | 2.2 | Iris-virginica |
| 133 | 6.3 | 2.8 | 5.1 | 1.5 | Iris-virginica |
| 134 | 6.1 | 2.6 | 5.6 | 1.4 | Iris-virginica |
| 135 | 7.7 | 3.0 | 6.1 | 2.3 | Iris-virginica |
| 136 | 6.3 | 3.4 | 5.6 | 2.4 | Iris-virginica |
| 137 | 6.4 | 3.1 | 5.5 | 1.8 | Iris-virginica |
| 138 | 6.0 | 3.0 | 4.8 | 1.8 | Iris-virginica |
| 139 | 6.9 | 3.1 | 5.4 | 2.1 | Iris-virginica |
| 140 | 6.7 | 3.1 | 5.6 | 2.4 | Iris-virginica |

```
141          6.9         3.1          5.1         2.3   Iris-virginica
142          5.8         2.7          5.1         1.9   Iris-virginica
143          6.8         3.2          5.9         2.3   Iris-virginica
144          6.7         3.3          5.7         2.5   Iris-virginica
145          6.7         3.0          5.2         2.3   Iris-virginica
146          6.3         2.5          5.0         1.9   Iris-virginica
147          6.5         3.0          5.2         2.0   Iris-virginica
148          6.2         3.4          5.4         2.3   Iris-virginica
149          5.9         3.0          5.1         1.8   Iris-virginica
```

#### 4.3.2 Get data types

[7]:
```
irisdata_df.info()
```

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

[8]:
```
irisdata_df.describe()
```

[8]:

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.054000 | 3.758667 | 1.198667 |
| std | 0.828066 | 0.433594 | 1.764420 | 0.763161 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

#### 4.3.3 Get data ranges with Boxplots

**Boxplots** can be used to explore the data ranges in the dataset. These also provide information about **outliers**.

[9]:
```
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.0})
sns.set_style("whitegrid")
#sns.set_style("white")

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

fn = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
cn = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
box1 = sns.boxplot(x = 'species', y = 'sepal_length',
                   data = irisdata_df, order = cn, ax = axs[0,0])
box2 = sns.boxplot(x = 'species', y = 'sepal_width',
                   data = irisdata_df, order = cn, ax = axs[0,1])
```

```
box3 = sns.boxplot(x = 'species', y = 'petal_length',
                   data = irisdata_df, order = cn, ax = axs[1,0])
box4 = sns.boxplot(x = 'species', y = 'petal_width',
                   data = irisdata_df,  order = cn, ax = axs[1,1])

# add some spacing between subplots
fig.tight_layout(pad=2.0)

plt.show()
```

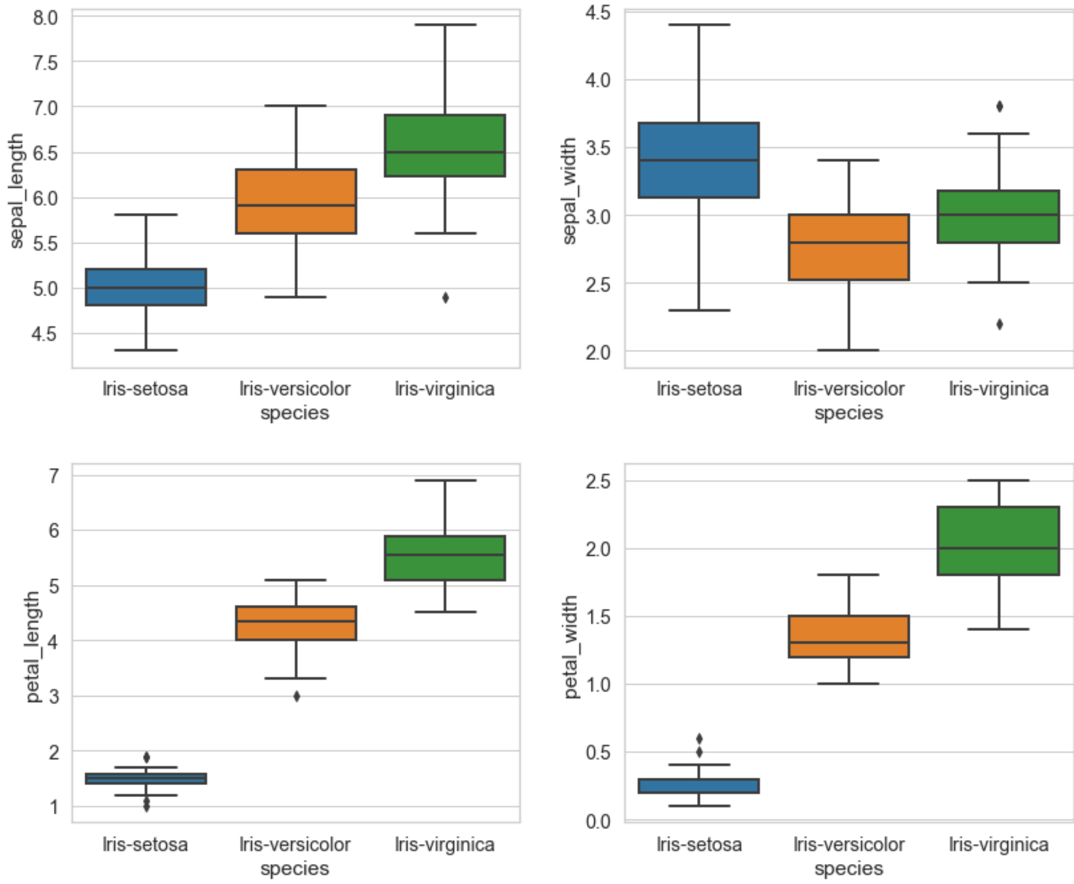



Figure 3: Boxplots used to explore the data ranges in the Iris dataset

## 4.4 Identify anomalies in the data sets

### 4.4.1 Find gaps in dataset

This section was inspired by Working with Missing Data in Pandas.

**Checking for missing values using `isnull()`** In order to check for missing values in Pandas DataFrame, we use the function `isnull()`. This function returns a dataframe of Boolean values which are True for **NaN values**.

[10]:
```
pd.set_option('display.max_rows', 40)
pd.set_option('display.min_rows', 30)
```

[11]:
```
irisdata_df.isnull()
```

[11]:
```
     sepal_length  sepal_width  petal_length  petal_width  species
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
5           False        False         False        False    False
6           False        False         False        False    False
7           False        False         False        False    False
8           False        False         False        False    False
9           False        False         False        False    False
10          False        False         False        False    False
11          False        False         False        False    False
12          False        False         False        False    False
13          False        False         False        False    False
14          False        False         False        False    False
..            …            …           …            …         …
135         False        False         False        False    False
136         False        False         False        False    False
137         False        False         False        False    False
138         False        False         False        False    False
139         False        False         False        False    False
140         False        False         False        False    False
141         False        False         False        False    False
142         False        False         False        False    False
143         False        False         False        False    False
144         False        False         False        False    False
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
```

Show only the gaps:

[12]:
```
irisdata_df_gaps = irisdata_df[irisdata_df.isnull().any(axis=1)]
irisdata_df_gaps
```

[12]:
```
Empty DataFrame
Columns: [sepal_length, sepal_width, petal_length, petal_width, species]
Index: []
```

Fine - this dataset seems to be complete :)

So let's look for something else for exercise: employes.csv

[13]:
```
# import data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")

employees_df
```

[13]:
```
  First Name  Gender  Start Date Last Login Time  Salary   Bonus %  \
0    Douglas    Male    8/6/1993        12:42 PM   97308   6945.00
1     Thomas    Male   3/31/1996         6:53 AM   61933      4.17
2      Maria  Female   4/23/1993        11:17 AM  130590  11858.00
3      Jerry    Male    3/4/2005         1:00 PM  138705      9.34
```

```
4         Larry    Male   1/24/1998         4:47 PM  101004   1389.00
5       Dennis    Male   4/18/1987         1:35 AM  115163  10125.00
6         Ruby  Female   8/17/1987         4:20 PM   65476  10012.00
7          NaN  Female   7/20/2015        10:43 AM   45906  11598.00
8       Angela  Female  11/22/2005         6:29 AM   95570  18523.00
9      Frances  Female    8/8/2002         6:51 AM  139852   7524.00
10      Louise  Female   8/12/1980         9:01 AM   63241  15132.00
11       Julie  Female  10/26/1997         3:19 PM  102508  12637.00
12     Brandon    Male   12/1/1980         1:08 AM  112807  17492.00
13        Gary    Male   1/27/2008        11:40 PM  109831   5831.00
14    Kimberly  Female   1/14/1999         7:13 AM   41426  14543.00
…          …     …          …             …      …       …
989    Stephen     NaN   7/10/1983         8:10 PM   85668   1909.00
990      Donna  Female  11/26/1982         7:04 AM   82871  17999.00
991     Gloria  Female   12/8/2014         5:08 AM  136709  10331.00
992      Alice  Female   10/5/2004         9:34 AM   47638  11209.00
993     Justin     NaN   2/10/1991         4:58 PM   38344   3794.00
994      Robin  Female   7/24/1987         1:35 PM  100765  10982.00
995       Rose  Female   8/25/2002         5:12 AM  134505  11051.00
996    Anthony    Male  10/16/2011         8:35 AM  112769  11625.00
997       Tina  Female   5/15/1997         3:53 PM   56450     19.04
998     George    Male   6/21/2013         5:47 PM   98874   4479.00
999      Henry     NaN  11/23/2014         6:09 AM  132483  16655.00
1000   Phillip    Male   1/31/1984         6:30 AM   42392  19675.00
1001   Russell    Male   5/20/2013        12:39 PM   96914   1421.00
1002     Larry    Male   4/20/2013         4:45 PM   60500  11985.00
1003    Albert    Male   5/15/2012         6:24 PM  129949  10169.00

     Senior Management                  Team
0                 True             Marketing
1                 True                   NaN
2                False               Finance
3                 True               Finance
4                 True       Client Services
5                False                 Legal
6                 True               Product
7                  NaN               Finance
8                 True           Engineering
9                 True  Business Development
10                True                   NaN
11                True                 Legal
12                True       Human Resources
13               False                 Sales
14                True               Finance
…                    …                     …
989              False                 Legal
990              False             Marketing
991               True               Finance
992              False       Human Resources
993              False                 Legal
994               True       Client Services
995               True             Marketing
996               True               Finance
997               True           Engineering
998               True             Marketing
999              False          Distribution
1000             False               Finance
```

```
1001              False                Product
1002              False   Business Development
1003               True                  Sales

[1004 rows x 8 columns]
```

Show only the gaps from this gappy dataset again:

[14]:
```
employees_df_gaps = employees_df[employees_df.isnull().any(axis=1)]
employees_df_gaps
```

[14]:
```
    First Name  Gender  Start Date Last Login Time  Salary   Bonus %  \
1       Thomas    Male   3/31/1996         6:53 AM   61933      4.17
7          NaN  Female   7/20/2015        10:43 AM   45906  11598.00
10      Louise  Female   8/12/1980         9:01 AM   63241  15132.00
20        Lois     NaN   4/22/1995         7:18 PM   64714   4934.00
22      Joshua     NaN    3/8/2012         1:58 AM   90816  18816.00
23         NaN    Male   6/14/2012         4:19 PM  125792   5042.00
25         NaN    Male   10/8/2012         1:12 AM   37076  18576.00
27       Scott     NaN   7/11/1991         6:58 PM  122367   5218.00
31       Joyce     NaN   2/20/2005         2:40 PM   88657  12752.00
32         NaN    Male   8/21/1998         2:27 PM  122340   6417.00
39         NaN    Male   1/29/2016         2:33 AM  122173   7797.00
41   Christine     NaN   6/28/2015         1:08 AM   66582  11308.00
49       Chris     NaN   1/24/1980        12:13 PM  113590   3055.00
51         NaN     NaN  12/17/2011         8:29 AM   41126  14009.00
53        Alan     NaN    3/3/2014         1:28 PM   40341  17578.00
..         …       …         …              …      …       …
916        Joe    Male   12/8/1998        10:28 AM  126120      1.02
927      Irene     NaN   2/28/1991        10:23 PM  135369      4.38
929        NaN  Female   8/23/2000         4:19 PM   95866  19388.00
941      Aaron     NaN   1/22/1986         7:39 PM   63126  18424.00
942       Mark     NaN    9/9/2006        12:27 PM   44836   2657.00
943      Ralph     NaN   7/28/1995         6:53 PM   70635   2147.00
949     Gerald     NaN   4/15/1989        12:44 PM   93712  17426.00
950        NaN  Female   9/15/1985         1:50 AM  133472  16941.00
951        NaN    Male   7/30/2012         3:07 PM  107351   5329.00
955        NaN  Female   9/14/2010         5:19 AM  143638   9662.00
965    Antonio     NaN   6/18/1989         9:37 PM  103050      3.05
976     Victor     NaN   7/28/2006         2:49 PM   76381  11159.00
989    Stephen     NaN   7/10/1983         8:10 PM   85668   1909.00
993     Justin     NaN   2/10/1991         4:58 PM   38344   3794.00
999      Henry     NaN  11/23/2014         6:09 AM  132483  16655.00

    Senior Management                  Team
1                True                   NaN
7                 NaN               Finance
10               True                   NaN
20               True                 Legal
22               True       Client Services
23                NaN                   NaN
25                NaN       Client Services
27              False                 Legal
31              False               Product
32                NaN                   NaN
39                NaN       Client Services
41               True  Business Development
49              False                 Sales
```

```
51                NaN                 Sales
53               True               Finance
..                ...                   ...
916             False                   NaN
927             False  Business Development
929               NaN                 Sales
941             False       Client Services
942             False       Client Services
943             False       Client Services
949              True          Distribution
950               NaN          Distribution
951               NaN             Marketing
955               NaN                   NaN
965             False                 Legal
976              True                 Sales
989             False                 Legal
993             False                 Legal
999             False          Distribution

[237 rows x 8 columns]
```

**Fill the missing values with `fillna()`** Now we are going to fill all the null (NaN) values in Gender column with *"No Gender"*.

**Attention:** We are doing that directly in this dataframe with `inplace = True` - we don't make a deep copy!

[15]:
```
# filling a null values using fillna()
employees_df["Gender"].fillna("No Gender", inplace = True)
employees_df
```

[15]:
```
    First Name     Gender  Start Date Last Login Time  Salary   Bonus %  \
0      Douglas       Male    8/6/1993        12:42 PM   97308   6945.00
1       Thomas       Male   3/31/1996         6:53 AM   61933      4.17
2        Maria     Female   4/23/1993        11:17 AM  130590  11858.00
3        Jerry       Male    3/4/2005         1:00 PM  138705      9.34
4        Larry       Male   1/24/1998         4:47 PM  101004   1389.00
5       Dennis       Male   4/18/1987         1:35 AM  115163  10125.00
6         Ruby     Female   8/17/1987         4:20 PM   65476  10012.00
7          NaN     Female   7/20/2015        10:43 AM   45906  11598.00
8       Angela     Female  11/22/2005         6:29 AM   95570  18523.00
9      Frances     Female    8/8/2002         6:51 AM  139852   7524.00
10      Louise     Female   8/12/1980         9:01 AM   63241  15132.00
11       Julie     Female  10/26/1997         3:19 PM  102508  12637.00
12     Brandon       Male   12/1/1980         1:08 AM  112807  17492.00
13        Gary       Male   1/27/2008        11:40 PM  109831   5831.00
14    Kimberly     Female   1/14/1999         7:13 AM   41426  14543.00
..         ...        ...         ...             ...     ...       ...
989    Stephen  No Gender   7/10/1983         8:10 PM   85668   1909.00
990      Donna     Female  11/26/1982         7:04 AM   82871  17999.00
991     Gloria     Female   12/8/2014         5:08 AM  136709  10331.00
992      Alice     Female   10/5/2004         9:34 AM   47638  11209.00
993     Justin  No Gender   2/10/1991         4:58 PM   38344   3794.00
994      Robin     Female   7/24/1987         1:35 PM  100765  10982.00
995       Rose     Female   8/25/2002         5:12 AM  134505  11051.00
996    Anthony       Male  10/16/2011         8:35 AM  112769  11625.00
997       Tina     Female   5/15/1997         3:53 PM   56450     19.04
```

```
998      George       Male   6/21/2013         5:47 PM   98874   4479.00
999       Henry  No Gender  11/23/2014         6:09 AM  132483  16655.00
1000    Phillip       Male   1/31/1984         6:30 AM   42392  19675.00
1001    Russell       Male   5/20/2013        12:39 PM   96914   1421.00
1002      Larry       Male   4/20/2013         4:45 PM   60500  11985.00
1003     Albert       Male   5/15/2012         6:24 PM  129949  10169.00

     Senior Management                 Team
0                 True            Marketing
1                 True                  NaN
2                False              Finance
3                 True              Finance
4                 True      Client Services
5                False                Legal
6                 True              Product
7                  NaN              Finance
8                 True          Engineering
9                 True  Business Development
10                True                  NaN
11                True                Legal
12                True      Human Resources
13               False                Sales
14                True              Finance
…                    …                    …
989              False                Legal
990              False            Marketing
991               True              Finance
992              False      Human Resources
993              False                Legal
994               True      Client Services
995               True            Marketing
996               True              Finance
997               True          Engineering
998               True            Marketing
999              False         Distribution
1000             False              Finance
1001             False              Product
1002             False  Business Development
1003              True                Sales

[1004 rows x 8 columns]
```

**Dropping missing values using `dropna()`** In order to drop null values from a dataframe, we use `dropna()` function. This function drops rows or columns of datasets with NaN values in different ways.

Default is to drop rows with at least 1 null value (NaN). Giving the parameter `how = 'all'` the function drops rows with all data missing or contain null values (NaN).

[16]:
```
# making a new dataframe with dropped NaN values
employees_df_dropped = employees_df.dropna(axis = 0, how ='any')
employees_df_dropped
```

[16]:
```
     First Name    Gender  Start Date Last Login Time  Salary   Bonus %  \
0       Douglas      Male    8/6/1993        12:42 PM   97308   6945.00
2         Maria    Female   4/23/1993        11:17 AM  130590  11858.00
3         Jerry      Male    3/4/2005         1:00 PM  138705      9.34
4         Larry      Male   1/24/1998         4:47 PM  101004   1389.00
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | Dennis | Male | 4/18/1987 | | 1:35 AM | 115163 | 10125.00 |
| 6 | Ruby | Female | 8/17/1987 | | 4:20 PM | 65476 | 10012.00 |
| 8 | Angela | Female | 11/22/2005 | | 6:29 AM | 95570 | 18523.00 |
| 9 | Frances | Female | 8/8/2002 | | 6:51 AM | 139852 | 7524.00 |
| 11 | Julie | Female | 10/26/1997 | | 3:19 PM | 102508 | 12637.00 |
| 12 | Brandon | Male | 12/1/1980 | | 1:08 AM | 112807 | 17492.00 |
| 13 | Gary | Male | 1/27/2008 | | 11:40 PM | 109831 | 5831.00 |
| 14 | Kimberly | Female | 1/14/1999 | | 7:13 AM | 41426 | 14543.00 |
| 15 | Lillian | Female | 6/5/2016 | | 6:09 AM | 59414 | 1256.00 |
| 16 | Jeremy | Male | 9/21/2010 | | 5:56 AM | 90370 | 7369.00 |
| 17 | Shawn | Male | 12/7/1986 | | 7:45 PM | 111737 | 6414.00 |
| … | … | … | … | … | … | … | |
| 989 | Stephen | No Gender | 7/10/1983 | | 8:10 PM | 85668 | 1909.00 |
| 990 | Donna | Female | 11/26/1982 | | 7:04 AM | 82871 | 17999.00 |
| 991 | Gloria | Female | 12/8/2014 | | 5:08 AM | 136709 | 10331.00 |
| 992 | Alice | Female | 10/5/2004 | | 9:34 AM | 47638 | 11209.00 |
| 993 | Justin | No Gender | 2/10/1991 | | 4:58 PM | 38344 | 3794.00 |
| 994 | Robin | Female | 7/24/1987 | | 1:35 PM | 100765 | 10982.00 |
| 995 | Rose | Female | 8/25/2002 | | 5:12 AM | 134505 | 11051.00 |
| 996 | Anthony | Male | 10/16/2011 | | 8:35 AM | 112769 | 11625.00 |
| 997 | Tina | Female | 5/15/1997 | | 3:53 PM | 56450 | 19.04 |
| 998 | George | Male | 6/21/2013 | | 5:47 PM | 98874 | 4479.00 |
| 999 | Henry | No Gender | 11/23/2014 | | 6:09 AM | 132483 | 16655.00 |
| 1000 | Phillip | Male | 1/31/1984 | | 6:30 AM | 42392 | 19675.00 |
| 1001 | Russell | Male | 5/20/2013 | | 12:39 PM | 96914 | 1421.00 |
| 1002 | Larry | Male | 4/20/2013 | | 4:45 PM | 60500 | 11985.00 |
| 1003 | Albert | Male | 5/15/2012 | | 6:24 PM | 129949 | 10169.00 |

| | Senior Management | Team |
|---|---|---|
| 0 | True | Marketing |
| 2 | False | Finance |
| 3 | True | Finance |
| 4 | True | Client Services |
| 5 | False | Legal |
| 6 | True | Product |
| 8 | True | Engineering |
| 9 | True | Business Development |
| 11 | True | Legal |
| 12 | True | Human Resources |
| 13 | False | Sales |
| 14 | True | Finance |
| 15 | False | Product |
| 16 | False | Human Resources |
| 17 | False | Product |
| … | … | … |
| 989 | False | Legal |
| 990 | False | Marketing |
| 991 | True | Finance |
| 992 | False | Human Resources |
| 993 | False | Legal |
| 994 | True | Client Services |
| 995 | True | Marketing |
| 996 | True | Finance |
| 997 | True | Engineering |
| 998 | True | Marketing |
| 999 | False | Distribution |
| 1000 | False | Finance |

```
1001             False                Product
1002             False   Business Development
1003              True                  Sales

[903 rows x 8 columns]
```

Finally we compare the sizes of dataframes so that we learn how many rows had at least 1 Null value.

[17]:
```
print("Old data frame length:", len(employees_df))
print("New data frame length:", len(employees_df_dropped))
print("Number of rows with at least 1 NaN value: ",
      (len(employees_df)-len(employees_df_dropped)))
```

```
Old data frame length: 1004
New data frame length: 903
Number of rows with at least 1 NaN value:  101
```

### 4.4.2 Find and remove duplicates in dataset

This section was inspired by: - How to Find Duplicates in Pandas DataFrame (With Examples) - How to Drop Duplicate Rows in a Pandas DataFrame

**Checking for duplicate values using `duplicated()`** In order to check for duplicate values in Pandas DataFrame, we use a function `duplicated()`. This function can be used in two ways: - find duplicate rows across **all columns** with `duplicateRows = df[df.duplicated()]` - find duplicate rows across **specific columns** `duplicateRows = df[df.duplicated(subset=['col1', 'col2'])]`

Find duplicate rows across **all columns**:

[18]:
```
# import (again) data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")
```

[19]:
```
# find duplicate rows across all columns
duplicateRows = employees_df[employees_df.duplicated()]
duplicateRows
```

[19]:
```
    First Name  Gender  Start Date Last Login Time  Salary  Bonus %  \
112      Karen  Female  11/30/1999         7:46 AM  102488  17653.0
127      Linda  Female   5/25/2000         5:45 PM  119009  12506.0
296    Brandon     NaN   11/3/1997         8:17 PM  121333  15295.0
580   Nicholas    Male    3/1/2013         9:26 PM  101036   2826.0

    Senior Management                  Team
112              True               Product
127              True  Business Development
296             False  Business Development
580              True       Human Resources
```

[20]:
```
# argument keep='last' displays the first duplicate rows instead of the last
duplicateRows = employees_df[employees_df.duplicated(keep='last')]
duplicateRows
```

[20]:
```
    First Name  Gender  Start Date Last Login Time  Salary  Bonus %  \
55       Karen  Female  11/30/1999         7:46 AM  102488  17653.0
92       Linda  Female   5/25/2000         5:45 PM  119009  12506.0
153    Brandon     NaN   11/3/1997         8:17 PM  121333  15295.0
442   Nicholas    Male    3/1/2013         9:26 PM  101036   2826.0
```

```
    Senior Management                  Team
55                True              Product
92                True  Business Development
153              False  Business Development
442               True       Human Resources
```

Find duplicate rows across **specific columns**:

[21]:
```
# identify duplicate rows across 'First Name' and 'Last Login Time' columns
duplicateRows = employees_df[employees_df.duplicated(
                    subset=['First Name', 'Last Login Time'])]
duplicateRows
```

[21]:
```
    First Name  Gender  Start Date Last Login Time  Salary  Bonus %  \
112      Karen  Female  11/30/1999         7:46 AM  102488  17653.0
127      Linda  Female   5/25/2000         5:45 PM  119009  12506.0
296    Brandon     NaN   11/3/1997         8:17 PM  121333  15295.0
577        NaN  Female   1/13/2009         1:01 PM  118736   7421.0
580   Nicholas    Male    3/1/2013         9:26 PM  101036   2826.0
632        NaN     NaN    9/2/1988        12:49 PM  147309   1702.0
881        NaN    Male    9/5/1980         7:36 AM  114896  13823.0
929        NaN  Female   8/23/2000         4:19 PM   95866  19388.0
934      Nancy  Female   9/10/2001        11:57 PM   85213   2386.0
973      Linda  Female    2/4/2010         8:49 PM   44486  17308.0

    Senior Management                  Team
112               True              Product
127               True  Business Development
296              False  Business Development
577                NaN       Client Services
580               True       Human Resources
632                NaN          Distribution
881                NaN       Client Services
929                NaN                 Sales
934               True             Marketing
973               True           Engineering
```

[22]:
```
# argument keep='last' displays the first duplicate rows instead of the last
duplicateRows = employees_df[employees_df.duplicated(
                    subset=['First Name', 'Last Login Time'], keep='last')]
duplicateRows
```

[22]:
```
    First Name  Gender  Start Date Last Login Time  Salary   Bonus %  \
23         NaN    Male   6/14/2012         4:19 PM  125792   5042.00
37       Linda  Female  10/19/1981         8:49 PM   57427   9557.00
55       Karen  Female  11/30/1999         7:46 AM  102488  17653.00
66       Nancy  Female  12/15/2012        11:57 PM  125250   2672.00
92       Linda  Female   5/25/2000         5:45 PM  119009  12506.00
153    Brandon     NaN   11/3/1997         8:17 PM  121333  15295.00
222        NaN  Female   6/17/1991        12:49 PM   71945      5.56
269        NaN    Male    2/4/2005         1:01 PM   40451  16044.00
442   Nicholas    Male    3/1/2013         9:26 PM  101036   2826.00
778        NaN  Female   6/18/2000         7:36 AM  106428  10867.00

    Senior Management                  Team
23                 NaN                  NaN
```

```
37          True        Client Services
55          True                Product
66          True   Business Development
92          True   Business Development
153        False   Business Development
222          NaN              Marketing
269          NaN           Distribution
442         True        Human Resources
778          NaN                    NaN
```

**Dropping duplicate values using `drop_duplicates()`** In order to drop duplicate values from a dataframe, we use `drop_duplicates()` function.

This function can be used in two ways: - remove duplicate rows across **all columns** with `df.drop_duplicates()` - find duplicate rows across **specific columns** `df.drop_duplicates(subset=['col1', 'col2'])`

**Attention:** We are doing that directly in this dataframe with `inplace = True` - we don't make a deep copy!

Remove duplicate rows across **all columns**:

[23]:
```
# remove duplicate rows across all columns
employees_df.drop_duplicates(inplace=True)
employees_df
```

[23]:
```
     First Name  Gender  Start Date Last Login Time  Salary   Bonus %  \
0       Douglas    Male    8/6/1993        12:42 PM   97308   6945.00
1        Thomas    Male   3/31/1996         6:53 AM   61933      4.17
2         Maria  Female   4/23/1993        11:17 AM  130590  11858.00
3         Jerry    Male    3/4/2005         1:00 PM  138705      9.34
4         Larry    Male   1/24/1998         4:47 PM  101004   1389.00
5        Dennis    Male   4/18/1987         1:35 AM  115163  10125.00
6          Ruby  Female   8/17/1987         4:20 PM   65476  10012.00
7           NaN  Female   7/20/2015        10:43 AM   45906  11598.00
8        Angela  Female  11/22/2005         6:29 AM   95570  18523.00
9       Frances  Female    8/8/2002         6:51 AM  139852   7524.00
10       Louise  Female   8/12/1980         9:01 AM   63241  15132.00
11        Julie  Female  10/26/1997         3:19 PM  102508  12637.00
12      Brandon    Male   12/1/1980         1:08 AM  112807  17492.00
13         Gary    Male   1/27/2008        11:40 PM  109831   5831.00
14     Kimberly  Female   1/14/1999         7:13 AM   41426  14543.00
…           …       …          …              …        …        …
989     Stephen     NaN   7/10/1983         8:10 PM   85668   1909.00
990       Donna  Female  11/26/1982         7:04 AM   82871  17999.00
991      Gloria  Female   12/8/2014         5:08 AM  136709  10331.00
992       Alice  Female   10/5/2004         9:34 AM   47638  11209.00
993      Justin     NaN   2/10/1991         4:58 PM   38344   3794.00
994       Robin  Female   7/24/1987         1:35 PM  100765  10982.00
995        Rose  Female   8/25/2002         5:12 AM  134505  11051.00
996     Anthony    Male  10/16/2011         8:35 AM  112769  11625.00
997        Tina  Female   5/15/1997         3:53 PM   56450     19.04
998      George    Male   6/21/2013         5:47 PM   98874   4479.00
999       Henry     NaN  11/23/2014         6:09 AM  132483  16655.00
1000    Phillip    Male   1/31/1984         6:30 AM   42392  19675.00
1001    Russell    Male   5/20/2013        12:39 PM   96914   1421.00
1002      Larry    Male   4/20/2013         4:45 PM   60500  11985.00
1003     Albert    Male   5/15/2012         6:24 PM  129949  10169.00
```

```
     Senior Management                  Team
0                 True             Marketing
1                 True                   NaN
2                False               Finance
3                 True               Finance
4                 True       Client Services
5                False                 Legal
6                 True               Product
7                  NaN               Finance
8                 True           Engineering
9                 True  Business Development
10                True                   NaN
11                True                 Legal
12                True       Human Resources
13               False                 Sales
14                True               Finance
…                    …                     …
989              False                 Legal
990              False             Marketing
991               True               Finance
992              False       Human Resources
993              False                 Legal
994               True       Client Services
995               True             Marketing
996               True               Finance
997               True           Engineering
998               True             Marketing
999              False          Distribution
1000             False               Finance
1001             False               Product
1002             False  Business Development
1003              True                 Sales

[1000 rows x 8 columns]
```

Remove duplicate rows across **specific columns**:

[24]:
```
# remove duplicate rows across 'First Name' and 'Last Login Time' columns
employees_df.drop_duplicates(
    subset=['First Name', 'Last Login Time'], keep='last', inplace=True)
employees_df
```

[24]:
```
     First Name  Gender  Start Date Last Login Time  Salary   Bonus %  \
0       Douglas    Male    8/6/1993        12:42 PM   97308   6945.00
1        Thomas    Male   3/31/1996         6:53 AM   61933      4.17
2         Maria  Female   4/23/1993        11:17 AM  130590  11858.00
3         Jerry    Male    3/4/2005         1:00 PM  138705      9.34
4         Larry    Male   1/24/1998         4:47 PM  101004   1389.00
5        Dennis    Male   4/18/1987         1:35 AM  115163  10125.00
6          Ruby  Female   8/17/1987         4:20 PM   65476  10012.00
7           NaN  Female   7/20/2015        10:43 AM   45906  11598.00
8        Angela  Female  11/22/2005         6:29 AM   95570  18523.00
9       Frances  Female    8/8/2002         6:51 AM  139852   7524.00
10       Louise  Female   8/12/1980         9:01 AM   63241  15132.00
11        Julie  Female  10/26/1997         3:19 PM  102508  12637.00
12      Brandon    Male   12/1/1980         1:08 AM  112807  17492.00
13         Gary    Male   1/27/2008        11:40 PM  109831   5831.00
```

```
14    Kimberly  Female   1/14/1999        7:13 AM   41426  14543.00
…          …       …          …          …        …     …
989    Stephen     NaN   7/10/1983        8:10 PM   85668   1909.00
990      Donna  Female  11/26/1982        7:04 AM   82871  17999.00
991     Gloria  Female   12/8/2014        5:08 AM  136709  10331.00
992      Alice  Female   10/5/2004        9:34 AM   47638  11209.00
993     Justin     NaN   2/10/1991        4:58 PM   38344   3794.00
994      Robin  Female   7/24/1987        1:35 PM  100765  10982.00
995       Rose  Female   8/25/2002        5:12 AM  134505  11051.00
996    Anthony    Male  10/16/2011        8:35 AM  112769  11625.00
997       Tina  Female   5/15/1997        3:53 PM   56450     19.04
998     George    Male   6/21/2013        5:47 PM   98874   4479.00
999      Henry     NaN  11/23/2014        6:09 AM  132483  16655.00
1000   Phillip    Male   1/31/1984        6:30 AM   42392  19675.00
1001   Russell    Male   5/20/2013       12:39 PM   96914   1421.00
1002     Larry    Male   4/20/2013        4:45 PM   60500  11985.00
1003    Albert    Male   5/15/2012        6:24 PM  129949  10169.00

     Senior Management                  Team
0                 True             Marketing
1                 True                   NaN
2                False               Finance
3                 True               Finance
4                 True       Client Services
5                False                 Legal
6                 True               Product
7                  NaN               Finance
8                 True           Engineering
9                 True  Business Development
10                True                   NaN
11                True                 Legal
12                True       Human Resources
13               False                 Sales
14                True               Finance
…                 …                 …
989              False                 Legal
990              False             Marketing
991               True               Finance
992              False       Human Resources
993              False                 Legal
994               True       Client Services
995               True             Marketing
996               True               Finance
997               True           Engineering
998               True             Marketing
999              False          Distribution
1000             False               Finance
1001             False               Product
1002             False  Business Development
1003              True                 Sales

[994 rows x 8 columns]
```

## 4.5 Avoidance of tendencies due to bias

The description of the Iris dataset says, that it consists of **50 samples** from **each of three species** of Iris (Iris setosa, Iris virginica and Iris versicolor), so there are **150 total samples**.

But how to prove it?

### 4.5.1 Count occurrences of unique values

To prove whether all possible classes included in the dataset and equally distributed, you can use the function `df.value_counts`.

Following parameters can be used for fine tuning: - `dropna=False` causes that NaN values are included - `normalize=True`: relative frequencies of the unique values are returned - `ascending=False`: sort resulting classes descending

[25]:
```
# import (again) data to dataframe from csv file
employees_df = pd.read_csv("./datasets/employees_edit.csv")
```

[26]:
```
# count unique values without missing values in a column,
# ordered descending and normalized
irisdata_df['species'].value_counts(ascending=False, dropna=False, normalize=True)
```

[26]:
```
Iris-setosa        0.333333
Iris-versicolor    0.333333
Iris-virginica     0.333333
Name: species, dtype: float64
```

[27]:
```
# count unique values and missing values in a column,
# ordered descending and not absolute values
employees_df['Team'].value_counts(ascending=False, dropna=False, normalize=False)
```

[27]:
```
Client Services         106
Business Development    103
Finance                 102
Marketing                98
Product                  96
Sales                    94
Engineering              92
Human Resources          92
Distribution             90
Legal                    88
NaN                      43
Name: Team, dtype: int64
```

### 4.5.2 Display Histogram

This section was inspired by: Pandas Histogram – DataFrame.hist().

**Histograms** represent **frequency distributions** graphically. This requires the separation of the data into classes (so-called **bins**).

These classes are represented in the histogram as rectangles of equal or variable width. The height of each rectangle then represents the (relative or absolute) **frequency density**.

[28]:
```
employees_df.hist(column=['Salary'])
plt.show()
```

[29]:
```
employees_df.hist(column='Salary', by='Gender')
plt.show()
```

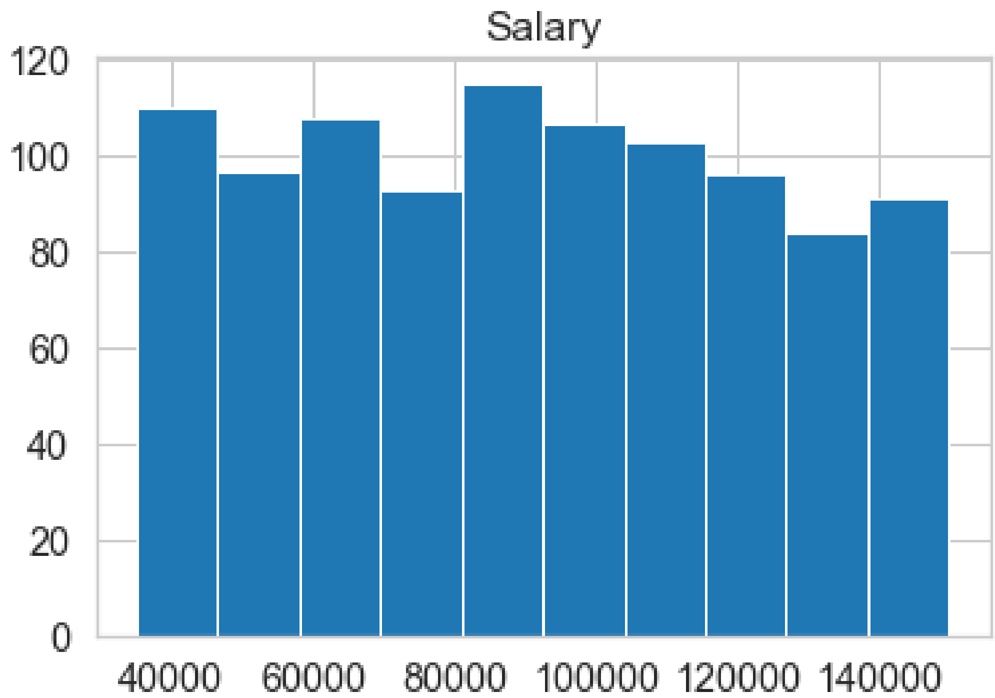



Figure 4:

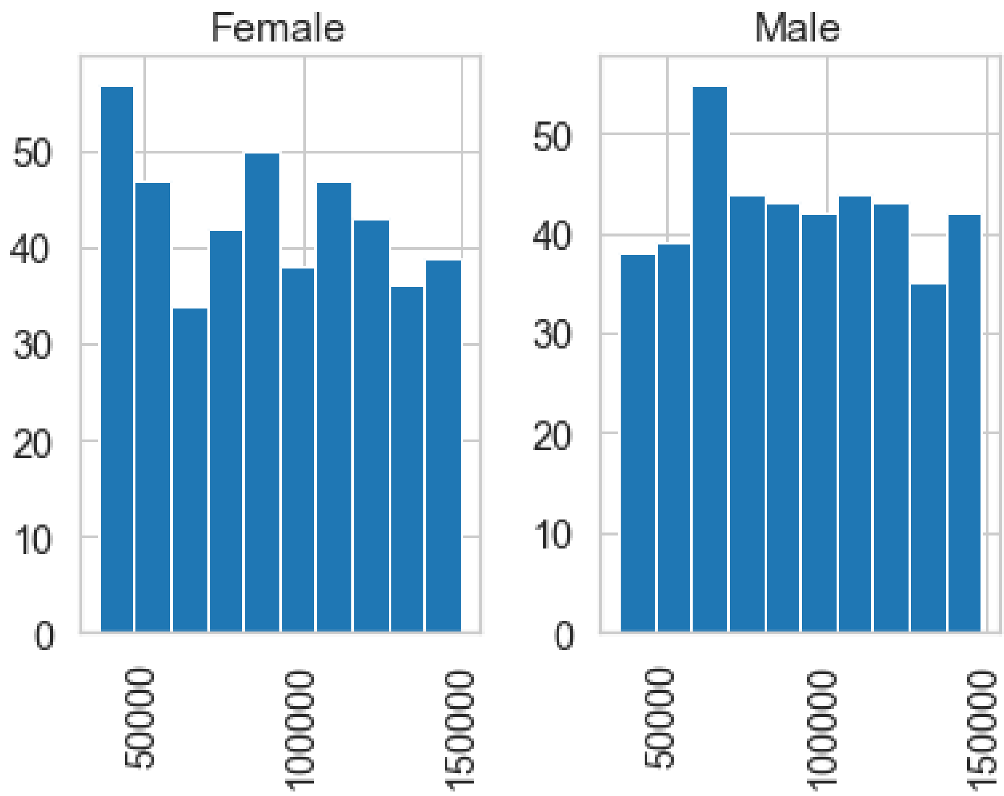



Figure 5:

## 4.6 First idea of correlations in data set

To get a rough idea of the **dependencies** and **correlations** in the data set, it can be helpful to visualize the whole dataset in a **correlation heatmap**. They show in a glance which variables are correlated, to what degree and in which direction.

Later, 2 particularly well correlated variables are selected from the data set and plotted in a **scatterplot**.

### 4.6.1 Visualise data with correlation heatmap

This section was inspired by How to Create a Seaborn Correlation Heatmap in Python?.

> **Correlation matrices** are an **essential tool of exploratory data analysis**. Correlation heatmaps contain the same information in a visually appealing way. What more: they show in a glance which variables are correlated, to what degree, in which direction, and alerts us to potential multicollinearity problems (source: ibidem).

**Simple correlation matrix** Because **string values can never be correlated**, the class names (species) have to be converted first:

[30]:
```python
# encoding the class column
irisdata_df_enc = irisdata_df.replace({"species":  {"Iris-setosa":0,
                                                    "Iris-versicolor":1,
                                                    "Iris-virginica":2}})

irisdata_df_enc
```

[30]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 0 |
| 5 | 5.4 | 3.9 | 1.7 | 0.4 | 0 |
| 6 | 4.6 | 3.4 | 1.4 | 0.3 | 0 |
| 7 | 5.0 | 3.4 | 1.5 | 0.2 | 0 |
| 8 | 4.4 | 2.9 | 1.4 | 0.2 | 0 |
| 9 | 4.9 | 3.1 | 1.5 | 0.1 | 0 |
| 10 | 5.4 | 3.7 | 1.5 | 0.2 | 0 |
| 11 | 4.8 | 3.4 | 1.6 | 0.2 | 0 |
| 12 | 4.8 | 3.0 | 1.4 | 0.1 | 0 |
| 13 | 4.3 | 3.0 | 1.1 | 0.1 | 0 |
| 14 | 5.8 | 4.0 | 1.2 | 0.2 | 0 |
| .. | … | … | … | … | … |
| 135 | 7.7 | 3.0 | 6.1 | 2.3 | 2 |
| 136 | 6.3 | 3.4 | 5.6 | 2.4 | 2 |
| 137 | 6.4 | 3.1 | 5.5 | 1.8 | 2 |
| 138 | 6.0 | 3.0 | 4.8 | 1.8 | 2 |
| 139 | 6.9 | 3.1 | 5.4 | 2.1 | 2 |
| 140 | 6.7 | 3.1 | 5.6 | 2.4 | 2 |
| 141 | 6.9 | 3.1 | 5.1 | 2.3 | 2 |
| 142 | 5.8 | 2.7 | 5.1 | 1.9 | 2 |
| 143 | 6.8 | 3.2 | 5.9 | 2.3 | 2 |
| 144 | 6.7 | 3.3 | 5.7 | 2.5 | 2 |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | 2 |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

```
[150 rows x 5 columns]
```

[31]:
```
irisdata_df_enc.corr()
```

[31]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| sepal_length | 1.000000 | -0.109369 | 0.871754 | 0.817954 | 0.782561 |
| sepal_width | -0.109369 | 1.000000 | -0.420516 | -0.356544 | -0.419446 |
| petal_length | 0.871754 | -0.420516 | 1.000000 | 0.962757 | 0.949043 |
| petal_width | 0.817954 | -0.356544 | 0.962757 | 1.000000 | 0.956464 |
| species | 0.782561 | -0.419446 | 0.949043 | 0.956464 | 1.000000 |

**Correlation heatmap** Choose the color sets from color map.

[32]:
```
# increase the size of the heatmap
plt.figure(figsize=(16, 6))

# store heatmap object in a variable to easily access it
# when you want to include more features (such as title)
# set the range of values to be displayed on the colormap from -1 to 1,
# and set 'annotation=True' to display the correlation values on the heatmap
heatmap = sns.heatmap(irisdata_df_enc.corr(), vmin=-1, vmax=1,
                      annot=True, cmap='PRGn_r')

# give a title to the heatmap
# 'pad=12' defines the distance of the title from the top of the heatmap
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=16)
plt.show()
```

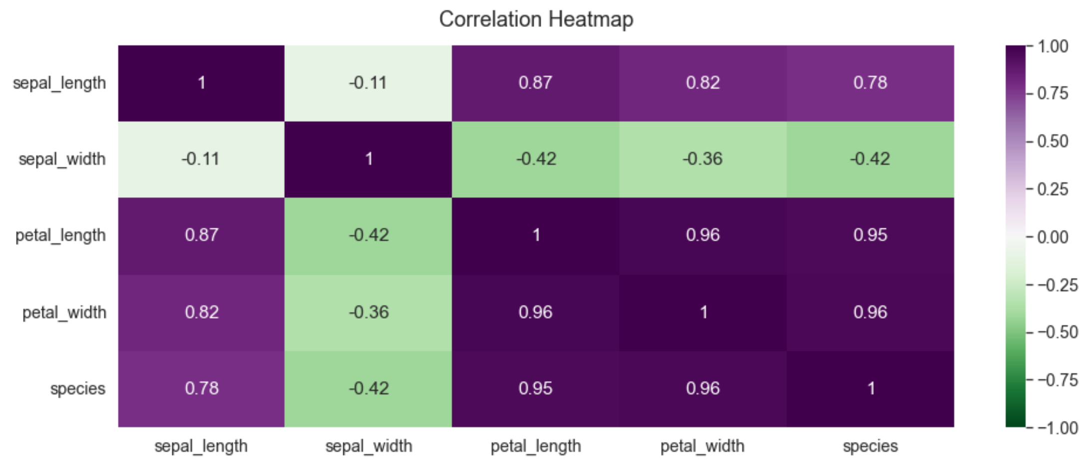



Figure 6:

**Triangle correlation heatmap** When looking at the correlation heatmaps above, you would not lose any information by **cutting** away half of it **along the diagonal** line marked by 1-s.

The **numpy** function `np.triu()` can be used to isolate the upper triangle of a matrix while turning all the values in the lower triangle into 0.

[33]:
```
import numpy as np

np.triu(np.ones_like(irisdata_df_enc.corr()))
```

[33]:
```
array([[1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1.],
       [0., 0., 0., 1., 1.],
       [0., 0., 0., 0., 1.]])
```

Use this mask to cut the heatmap along the diagonal:

[34]:
```
plt.figure(figsize=(16, 6))

# define the mask to set the values in the upper triangle to 'True'
mask = np.triu(np.ones_like(irisdata_df_enc.corr(), dtype=bool))

heatmap = sns.heatmap(irisdata_df_enc.corr(), mask=mask,
                      vmin=-1, vmax=1, annot=True, cmap='PRGn_r')

heatmap.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18}, pad=16)
plt.show()
```

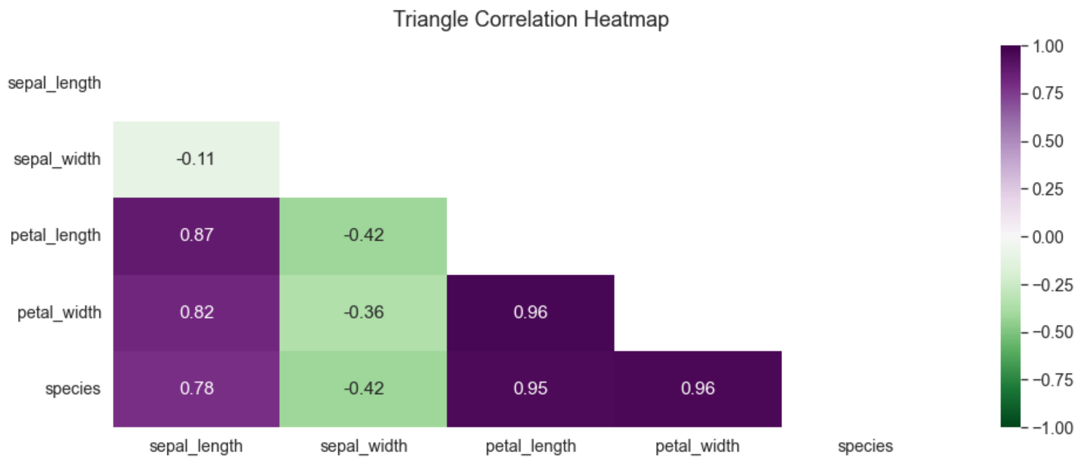



Figure 7:

As a result from the **heatmaps** we can see, that the shape of the **petals** are the **most correlationed columns** (0.96) with the **type of flowers** (species classes).

Somewhat lower correlates **sepal length** with **petal length** (0.87).

### 4.6.2 Visualise data with scatter plot

In the following, Seaborn is applied which is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

To investigate whether there are dependencies (e.g. correlations) in `irisdata_df` between individual variables in the data set, it is advisable to plot them in a **scatter plot**.

[69]:
```
# There are five preset seaborn themes: darkgrid, whitegrid, dark, white, and ticks.
sns.set_style("whitegrid")
# set scale of fonts
sns.set_context("notebook", font_scale=1.3, rc={"lines.linewidth": 2.5})

# 'sepal_length', 'petal_length' are iris feature data
# 'height' used to define height of graph
# 'hue' stores the class/label of iris dataset
sns.FacetGrid(irisdata_df, hue ="species",
```

```
                    height = 7).map(plt.scatter,
                                    'petal_width',
                                    'petal_length').add_legend()


plt.title('Scatterplot of petal length and width')
plt.show()
```

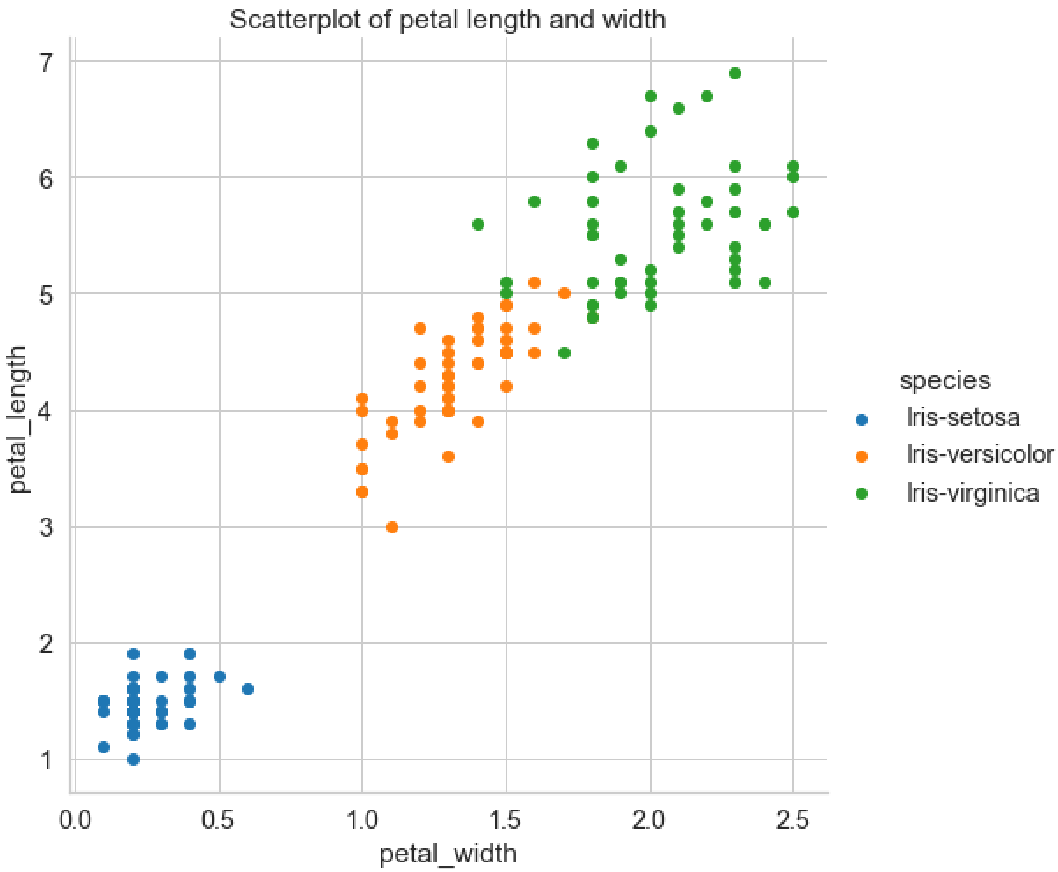



Figure 8:

#### 4.6.3 Visualise data with pairs plot

For systematic investigation of dependencies, all variables (each against each) are plotted in separate scatter plots.

With this so called **pairs plot** it is possible to see both **relationships** between two variables and **distribution** of single variables.

This function will create a grid of Axes such that **each numeric variable** in `irisdata_df` will by shared in the y-axis across a single row and in the x-axis across a single column.

[36]:
```
sns.set_style("white")
g = sns.pairplot(irisdata_df, diag_kind="kde", hue='species',
                 palette='Dark2', height=2.5)

g.map_lower(sns.kdeplot, levels=4, color=".2")

plt.show()
```

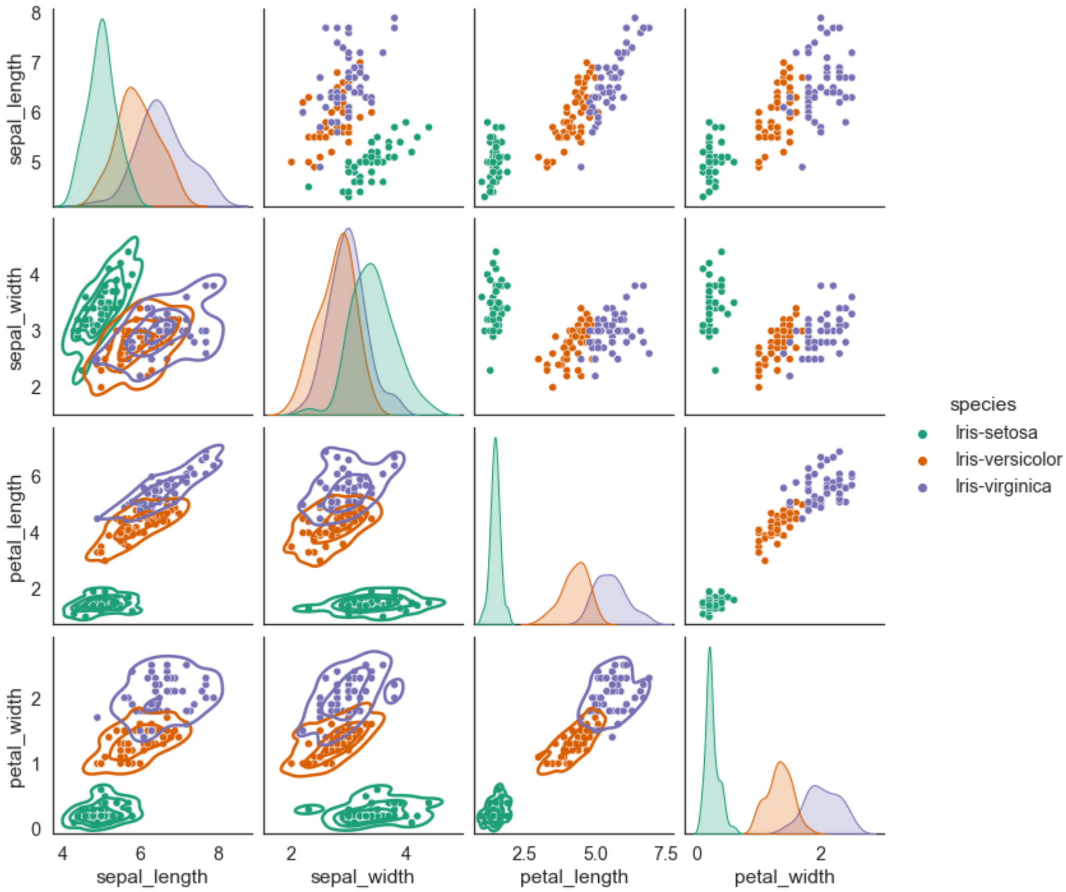



Figure 9:

# 5 STEP 2: Prepare the data

Through the intensive exploration of the data in Step 1 (STEP 1: Exploring the data), we know that special **preparation** of the data is **not necessary**. The values are **complete** and **without gaps** and there are **no duplicates**. The values are in similar ranges, which **does not require normalization** of the data.

Furthermore, we know that the **classes** are very **evenly distributed** and thus bias tendencies should be avoided.

# 6 STEP 3: Classify by support vector classifier - SVC

## 6.1 Operating principal

> Support Vectors Classifier tries to **find the best hyperplane to separate** the different classes by maximizing the distance between sample points and the hyperplane (source: In Depth: Parameter tuning for SVC).

The figure **??** shows the operating principal of the SVC algorithm: the hyperplanes *H1* till *H4* (left graphic) do separate the classes. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier (source: Support-vector machine).

The right graphic shows the optimal hyperplane characterized by maximising the margin between the classes. The perpendicular distance of the closest data points to the hyperplane determines their position and orientation. These perpendicular distances are the **support vectors** of the hyperplane - this is how the algorithm got its name.

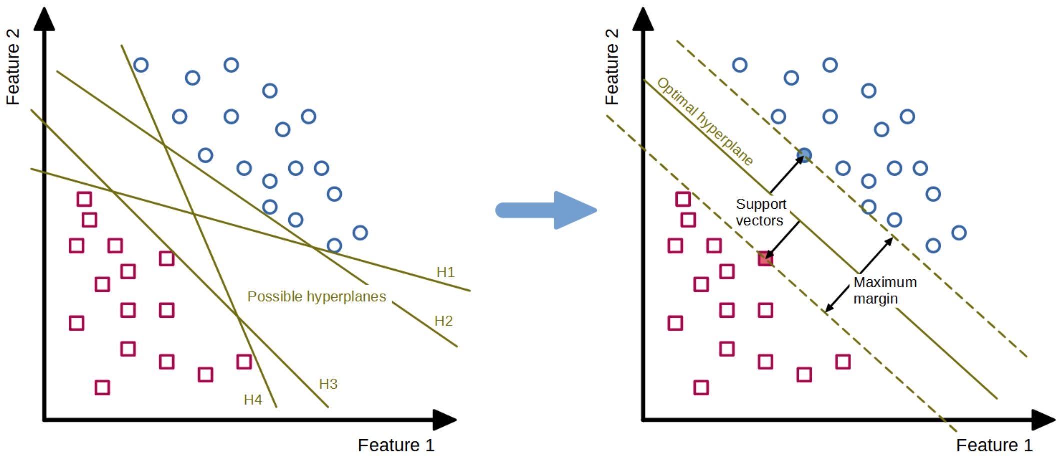



Figure 10: Support Vectors Classifiers (SVC) separate the data points in classes by finding the best hyperplane

## 6.2 Split the dataset

In the next very important step, the dataset is split into **2 subsets**: a **training dataset** and a **test dataset**. As the names suggest, the training dataset is used to train the ML algorithm. The test data set is then used to check the quality of the trained ML algorithm (here the **recognition rate**). For this purpose, the **class labels** are **removed** from the training data set - after all, these are to be predicted.

Typically, the **test dataset** should contain about **20%** of the entire dataset.

[52]:
```
from sklearn.model_selection import train_test_split

X = irisdata_df.drop('species', axis=1)
y = irisdata_df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

For training, do not use only the variables that correlate best with each other, but all of them.

Otherwise, the result of the prediction would be significantly worse. Maybe this is already an indication of **overfitting** of the ML model.

[38]:
```
# DO NOT USE THIS!!
X_train, X_test, y_train, y_test = train_test_split(X[['sepal_length',
                                                       'sepal_width']],
                                                    y, test_size = 0.20)
```

## 6.3 Create the SVM model

In this step we create the SVC model and fit it to our training data.

[53]:
```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)

# fit the model for the data
classifier.fit(X_train, y_train)
```

[53]:
```
SVC(kernel='linear', random_state=0)
```

### 6.4 Make predictions

[54]:
```
y_pred = classifier.predict(X_test)
#X_test
```

## 7 STEP 4: Evaluate the results - metrics

And finally for checking the accuracy of the model, the **confusion matrix** is used for the **cross validation**.

By using the function `sklearn.metrics.confusion_matrix()` a confusion matrix of the true digit values versus the predicted digit values is plotted.

### 7.1 Textual confusion matrix

[55]:
```
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[14  0  0]
 [ 0  9  1]
 [ 0  0  6]]
```

### 7.2 Colored confusion matrix

The function `sklearn.metrics.ConfusionMatrixDisplay()` plots a colored confusion matrix.

[67]:
```
sns.set_style("white")

# print colored confusion matrix
cm_colored = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

#cm_colored.figure_.suptitle("Confusion Matrix")
cm_colored.figure_.set_figwidth(8)
cm_colored.figure_.set_figheight(7)

cm_colored.confusion_matrix

# save figure as PNG
plt.tight_layout()
plt.savefig('images/confusion_matrix.png', dpi=150, pad_inches=5)
plt.show()
```

[43]:
```
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator = classifier, X = X_train,
                             y = y_train, cv = 10)

print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 79.17 %
Standard Deviation: 6.72 %
```

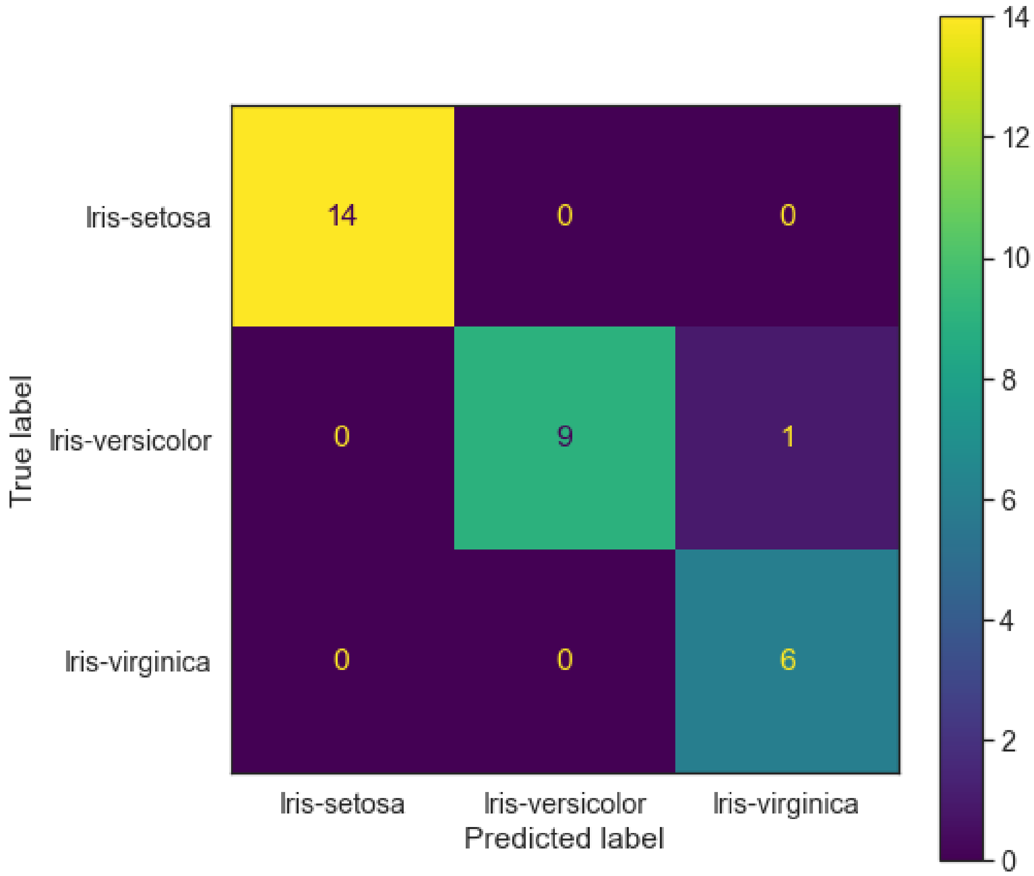



Figure 11:

# 8 STEP 5: Vary parameters

This section was inspired by In Depth: Parameter tuning for SVC

In this section, the 4 SVC parameters `kernel`, `gamma`, `C` and `degree` will be introduced one by one. Furthermore, their influence on the classification result by varying these single parameters will be shown.

**Disclaimer:** In order to show the effects of varying the individual parameters in 2D graphs, only the best correlating variables `petal_length` and `petal_width` are used to train the SVC.

## 8.1 Prepare dataset

[44]:
```
# import iris dataset again
irisdata_df = pd.read_csv('./datasets/IRIS_flower_dataset_kaggle.csv')

# encode the class column from class strings to integer equivalents
irisdata_df_enc = irisdata_df.replace({"species":  {"Iris-setosa":0,
                                                    "Iris-versicolor":1,
                                                    "Iris-virginica":2}})
irisdata_df_enc
```

[44]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 0 |

```
4       5.0     3.6     1.4     0.2     0
5       5.4     3.9     1.7     0.4     0
6       4.6     3.4     1.4     0.3     0
7       5.0     3.4     1.5     0.2     0
8       4.4     2.9     1.4     0.2     0
9       4.9     3.1     1.5     0.1     0
10      5.4     3.7     1.5     0.2     0
11      4.8     3.4     1.6     0.2     0
12      4.8     3.0     1.4     0.1     0
13      4.3     3.0     1.1     0.1     0
14      5.8     4.0     1.2     0.2     0
..      …       …       …       …       …
135     7.7     3.0     6.1     2.3     2
136     6.3     3.4     5.6     2.4     2
137     6.4     3.1     5.5     1.8     2
138     6.0     3.0     4.8     1.8     2
139     6.9     3.1     5.4     2.1     2
140     6.7     3.1     5.6     2.4     2
141     6.9     3.1     5.1     2.3     2
142     5.8     2.7     5.1     1.9     2
143     6.8     3.2     5.9     2.3     2
144     6.7     3.3     5.7     2.5     2
145     6.7     3.0     5.2     2.3     2
146     6.3     2.5     5.0     1.9     2
147     6.5     3.0     5.2     2.0     2
148     6.2     3.4     5.4     2.3     2
149     5.9     3.0     5.1     1.8     2

[150 rows x 5 columns]
```

[45]:

```
# copy only 2 feature columns
# and convert pandas dataframe to numpy array
X = irisdata_df_enc[['petal_length', 'petal_width']].to_numpy(copy=True)
#X = irisdata_df_enc[['sepal_length', 'sepal_width']].to_numpy(copy=True)
#X
```

[46]:

```
# convert pandas dataframe to numpy array
# and get a flat 1D copy of 2D numpy array
y = irisdata_df_enc[['species']].to_numpy(copy=True).flatten()
#y
```

## 8.2 Plotting function

This function helps to visualize the modifications by varying the individual SVC parameters.

[47]:

```
def plotSVC(title, xlabel, ylabel):
    # create a mesh to plot in
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # prevent division by zero
    if x_min == 0.0:
        x_min = 0.1

    h = (x_max / x_min)/1000
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
    plt.subplot(1, 1, 1)
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.6)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xlim(xx.min(), xx.max())
    plt.title(title)
    plt.show()
```

### 8.3 Vary `kernel` parameter

The `kernel` parameter selects the type of hyperplane that is used to separate the data. Using `linear` (linear classifier) kernel will use a linear hyperplane (a line in the case of 2D data). The `rbf` (radial basis function kernel) and `poly` (polynomial kernel) kernel use non linear hyperplanes.

[48]:
```
kernels = ['linear', 'rbf', 'poly']

xlabel = 'Petal length'
ylabel = 'Petal width'

for kernel in kernels:
    svc = svm.SVC(kernel=kernel).fit(X, y)
    plotSVC('kernel = ' + str(kernel), xlabel, ylabel)
```

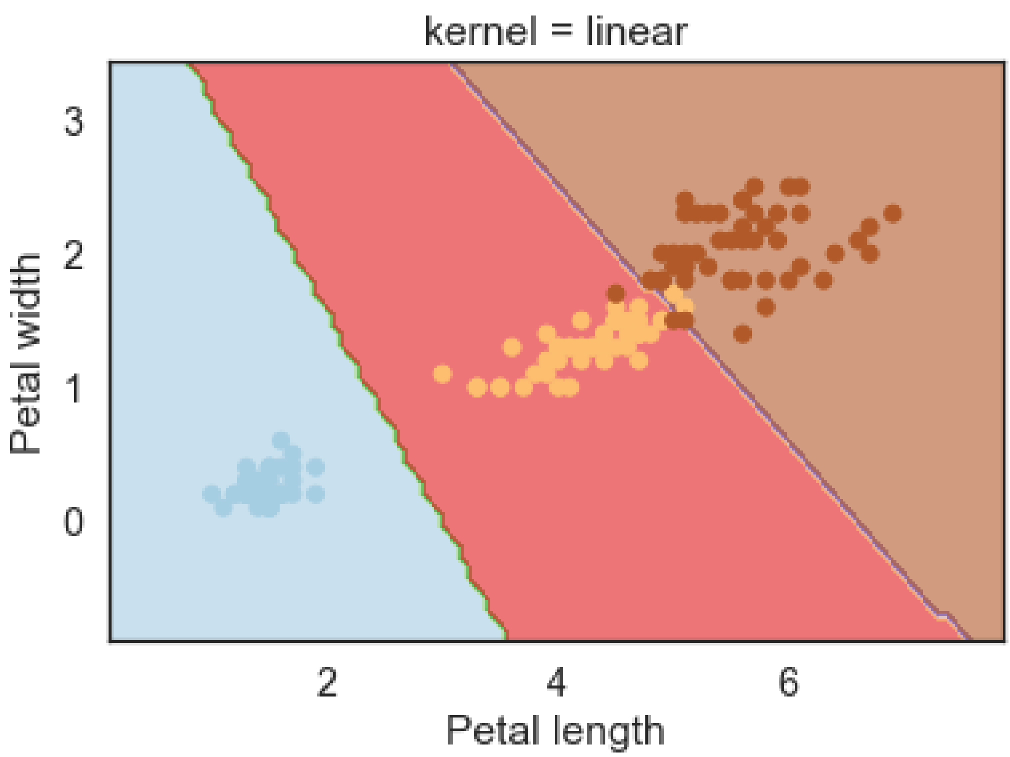



Figure 12:

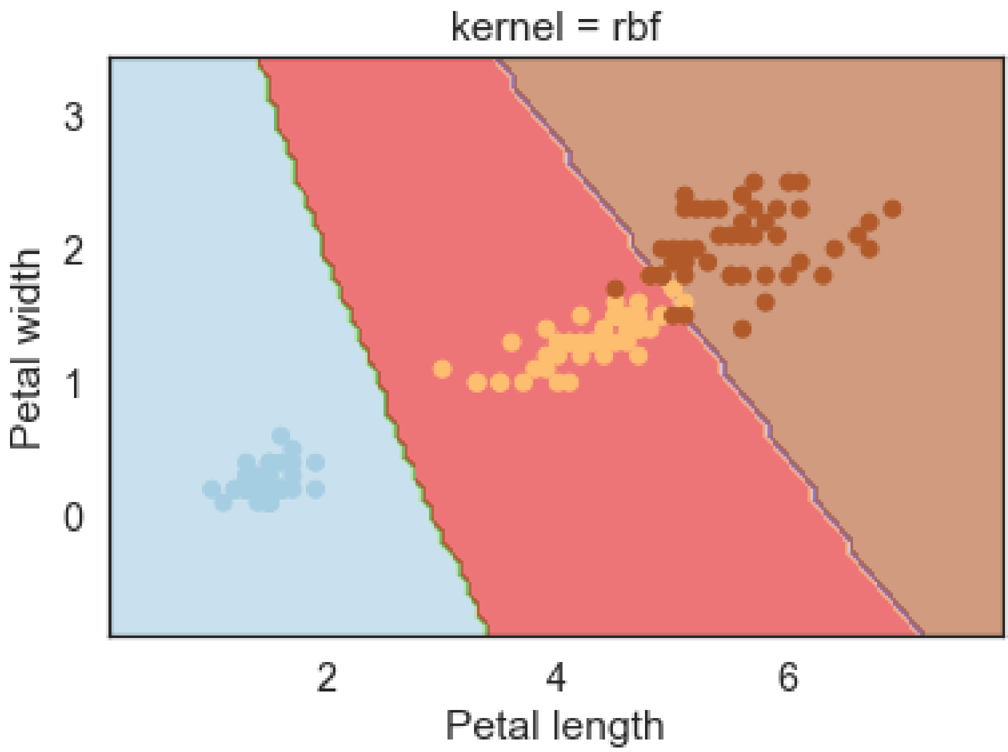



Figure 13:

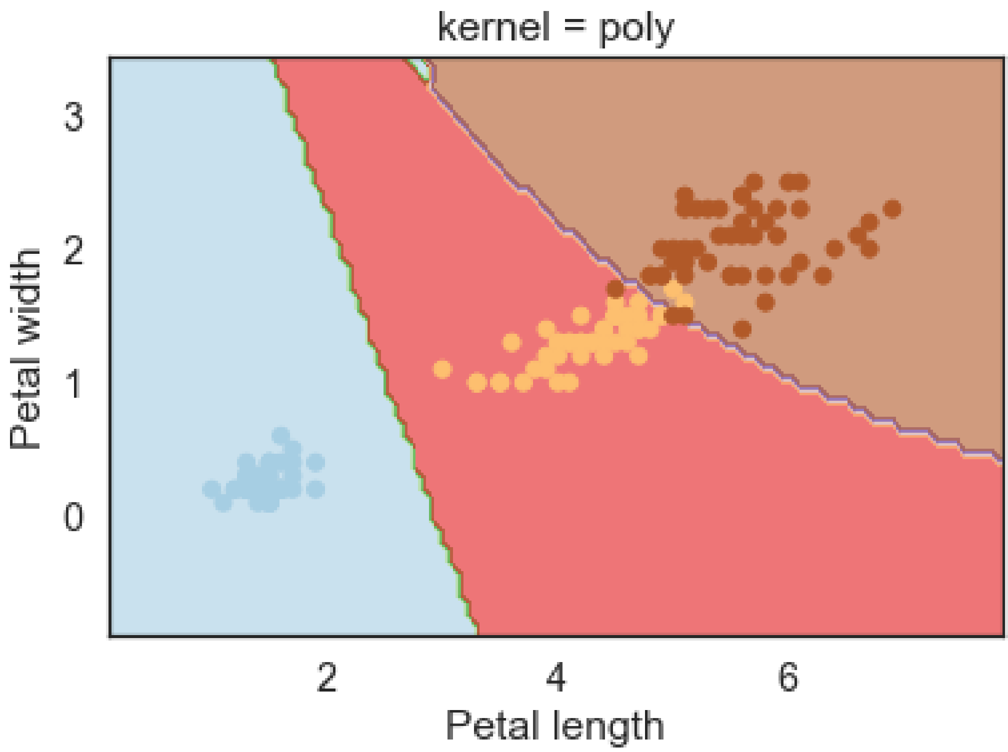



Figure 14:

## 8.4 Vary gamma parameter

The `gamma` parameter is used for non linear hyperplanes. The higher the `gamma` value it tries to exactly fit the training data set.

As we can see, increasing `gamma` leads to **overfitting** as the classifier tries to perfectly fit the training data.

[49]:
```
gammas = [0.1, 1, 10, 100, 200]

xlabel = 'Petal length'
ylabel = 'Petal width'

for gamma in gammas:
    svc = svm.SVC(kernel='rbf', gamma=gamma).fit(X, y)
    plotSVC('gamma = ' + str(gamma), xlabel, ylabel)
```

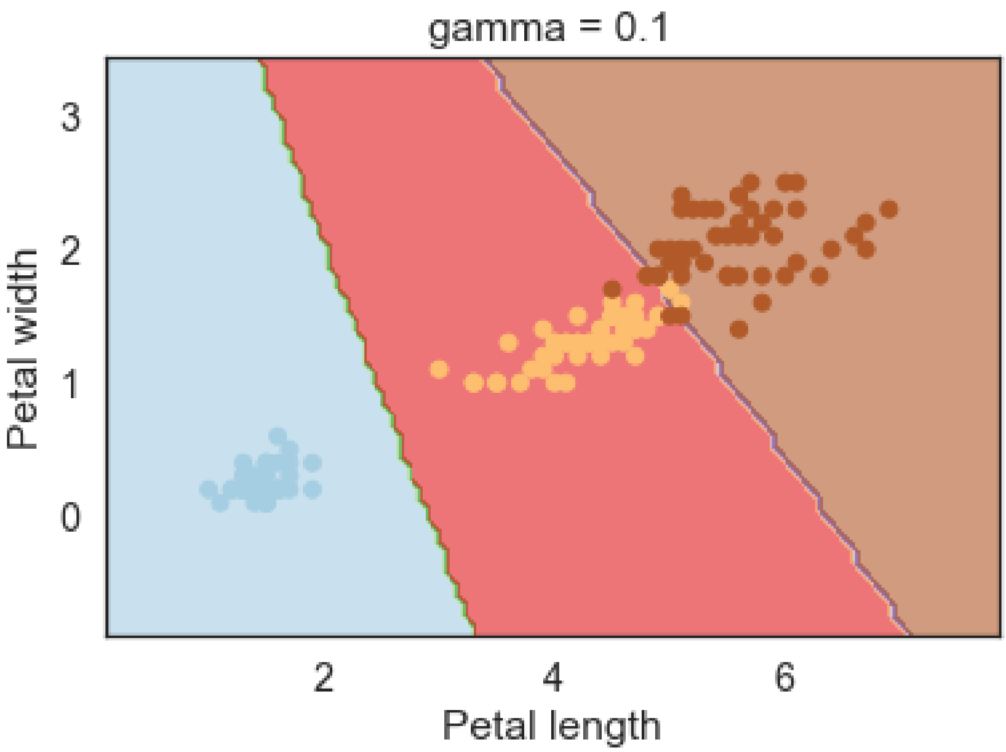



Figure 15:

## 8.5 Vary C parameter

The `C` parameter is the **penalty** of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

But be careful: to high `C` values may lead to **overfitting** the training data.

[50]:
```
cs = [0.1, 1, 10, 100, 1000, 10000]

xlabel = 'Petal length'
ylabel = 'Petal width'

for c in cs:
    svc = svm.SVC(kernel='rbf', C=c).fit(X, y)
```

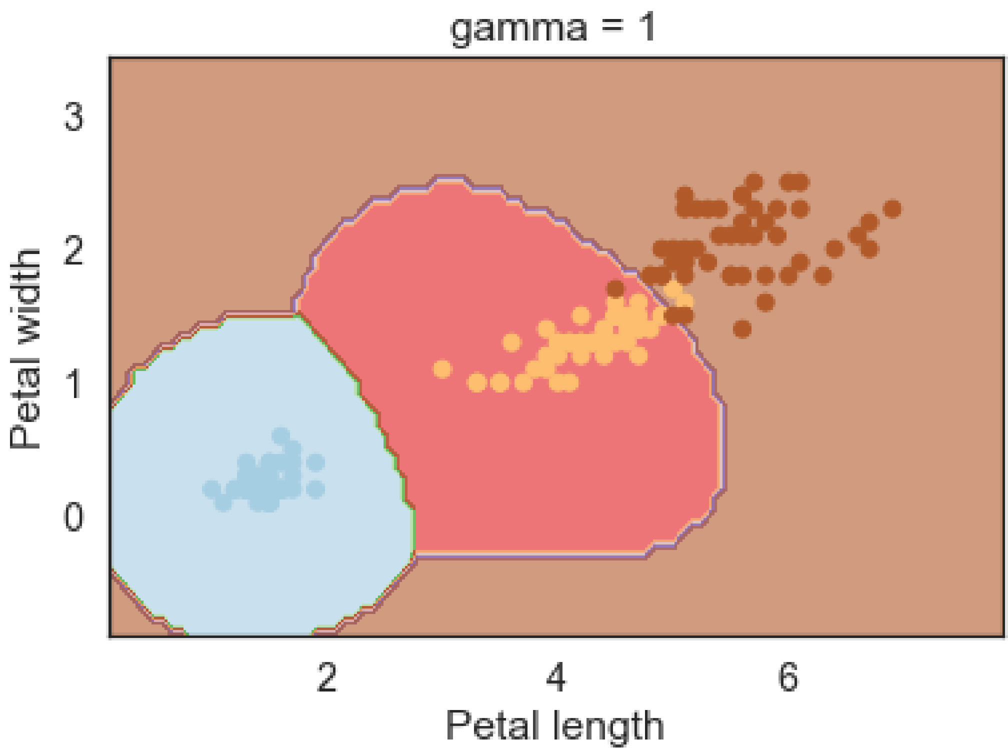



Figure 16:

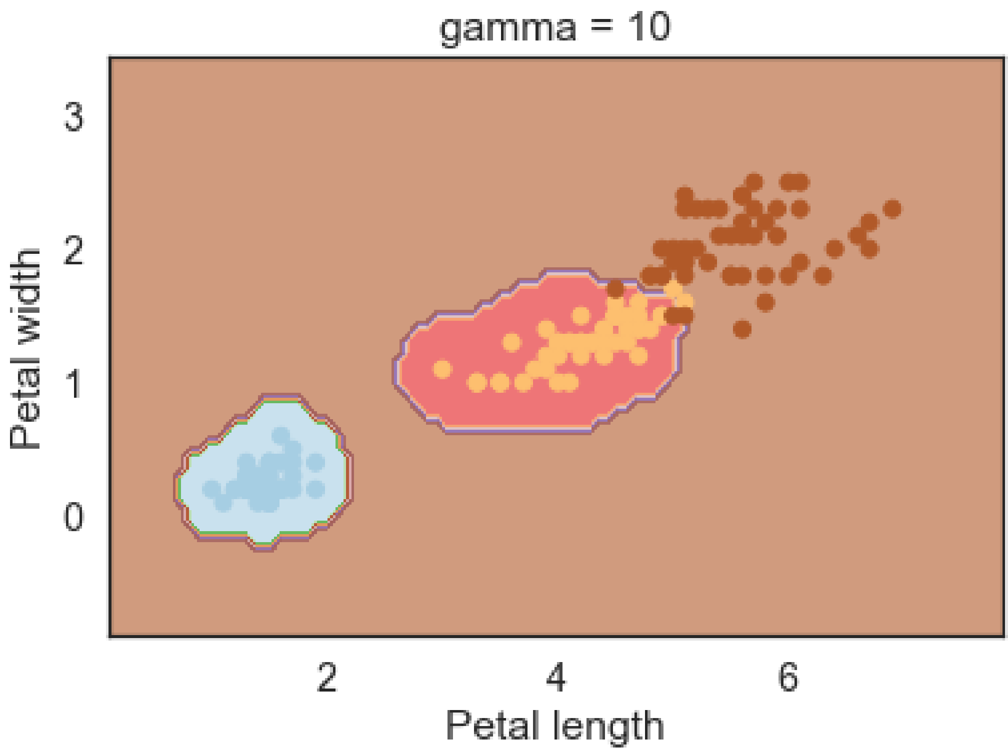



Figure 17:

```
plotSVC('C = ' + str(c), xlabel, ylabel)
```

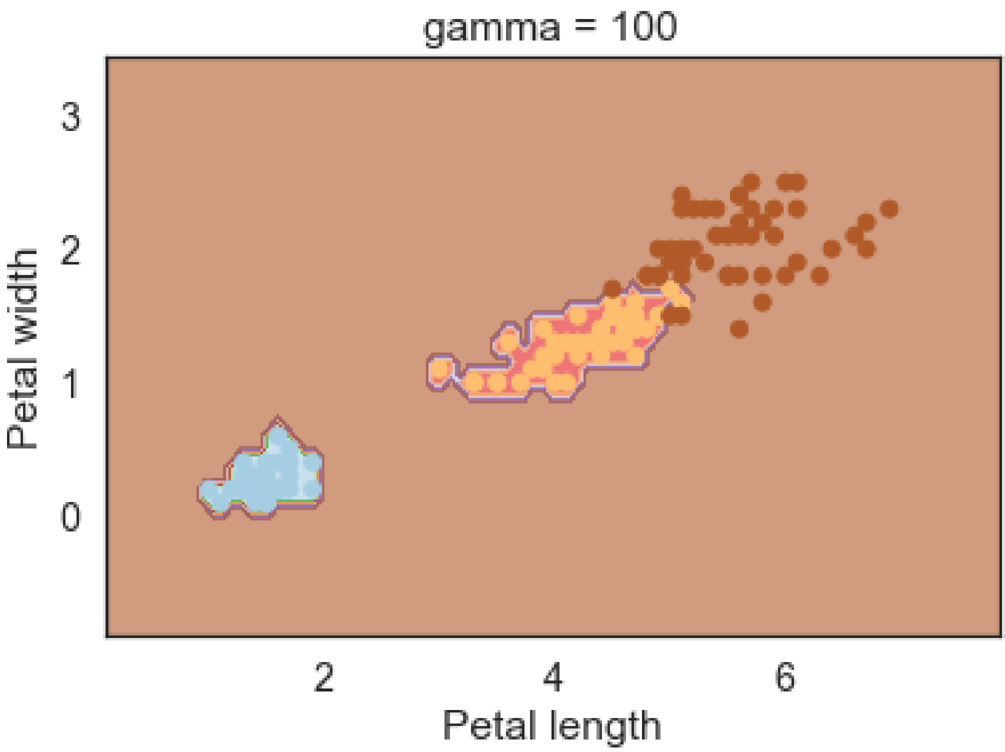



Figure 18:

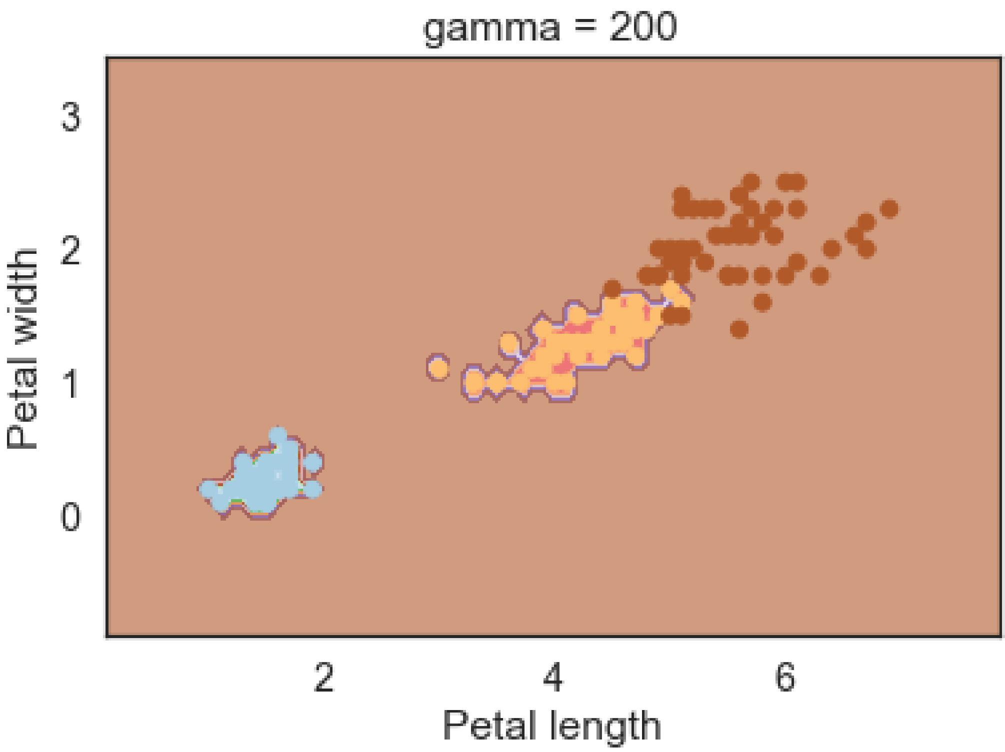



Figure 19:

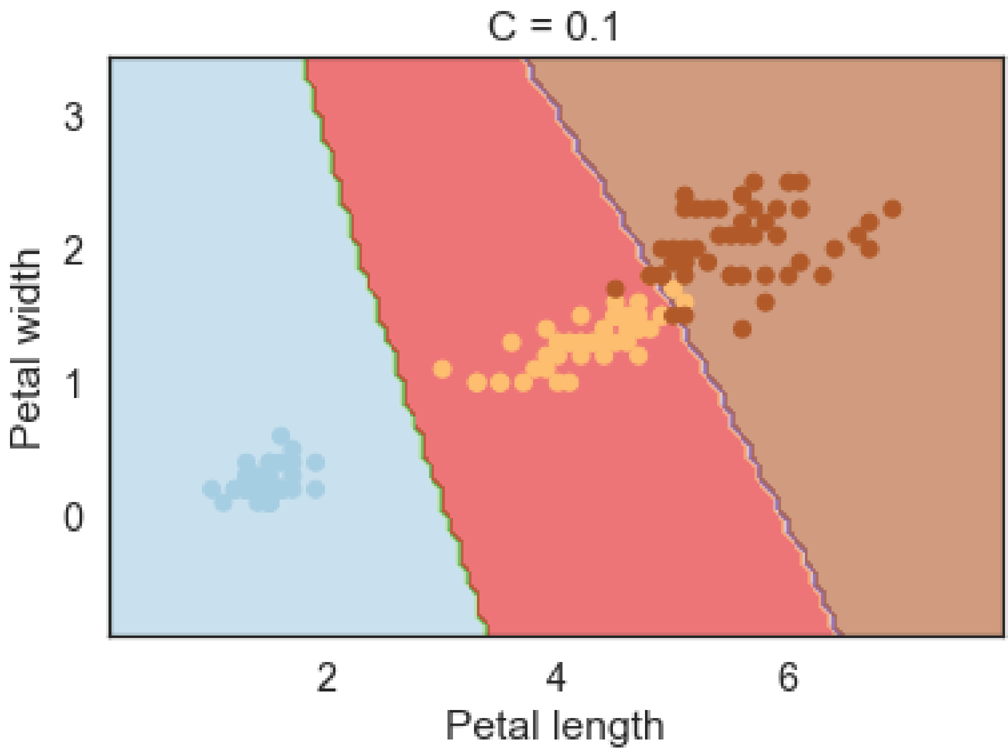



Figure 20:

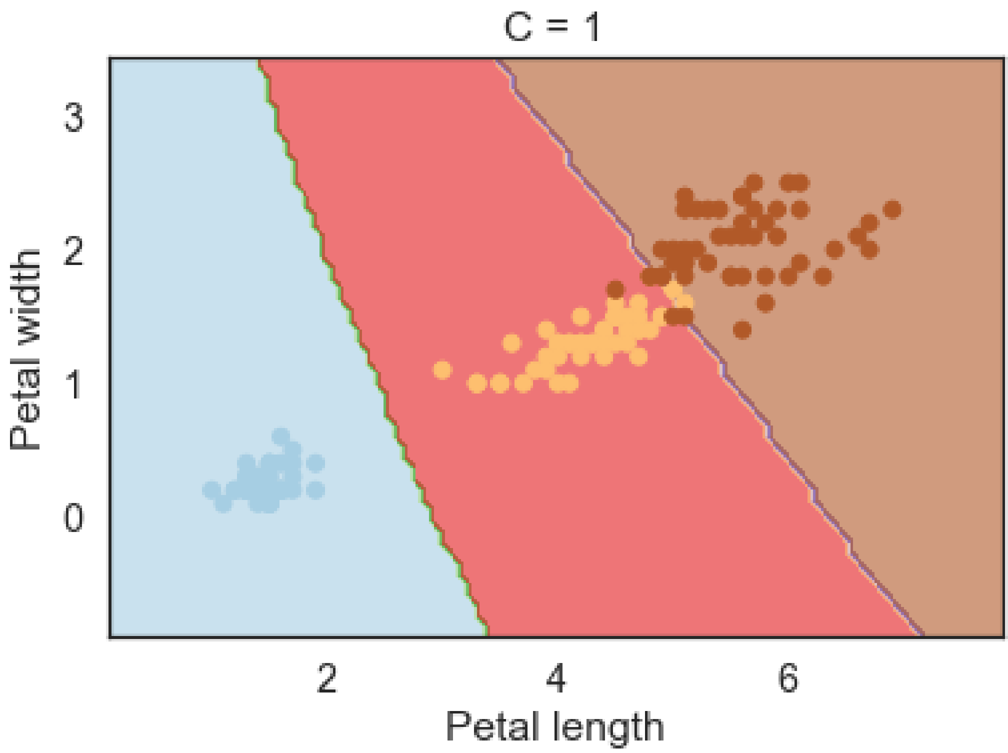



Figure 21:

## 8.6 Vary `degree` parameter

The `degree` parameter is used when the `kernel` is set to `poly`. It's basically the **degree of the polynomial** used to find the hyperplane to split the data.

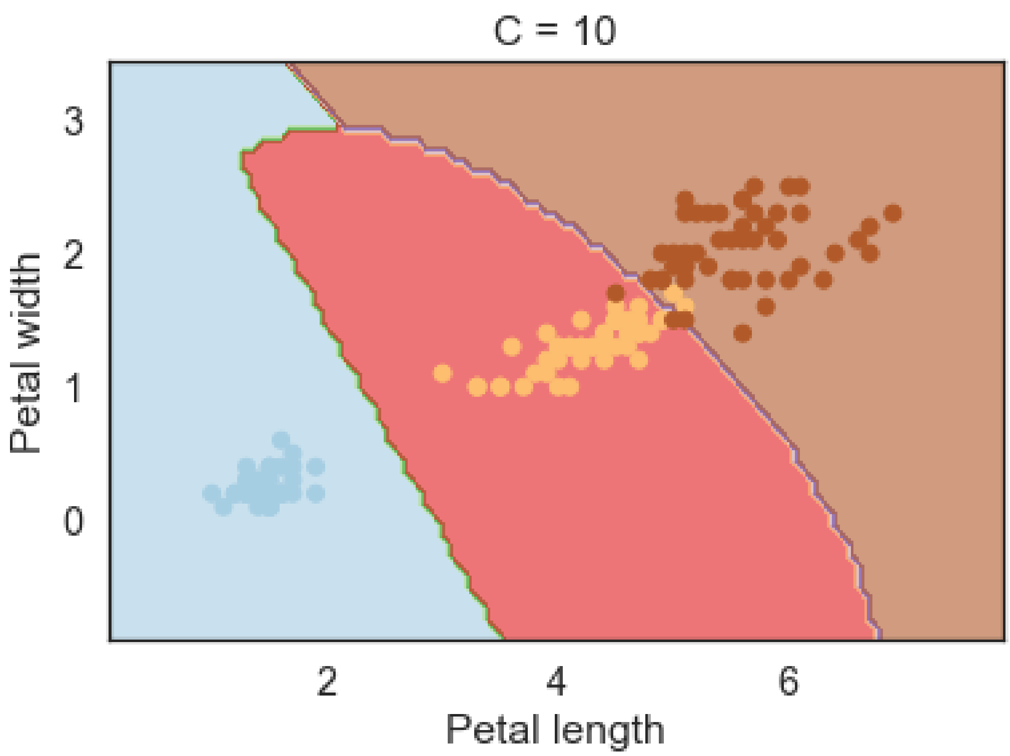



Figure 22:

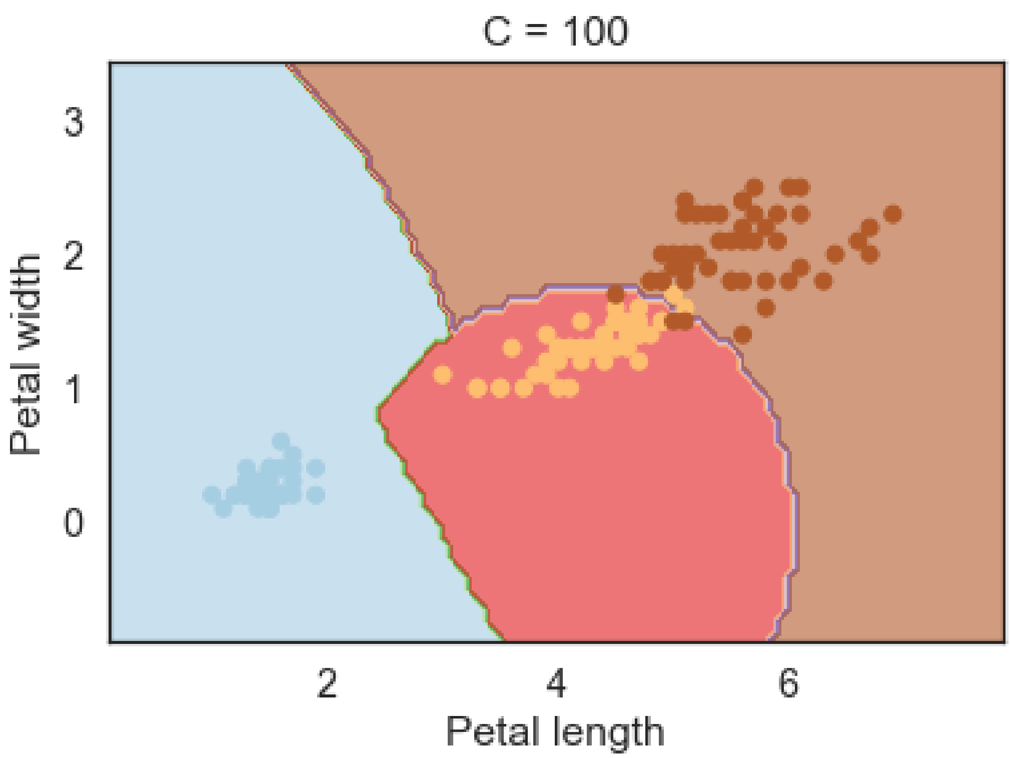



Figure 23:

Using `degree = 1` is the same as using a `linear` kernel. Also, increasing this parameters leads to **higher training times**.

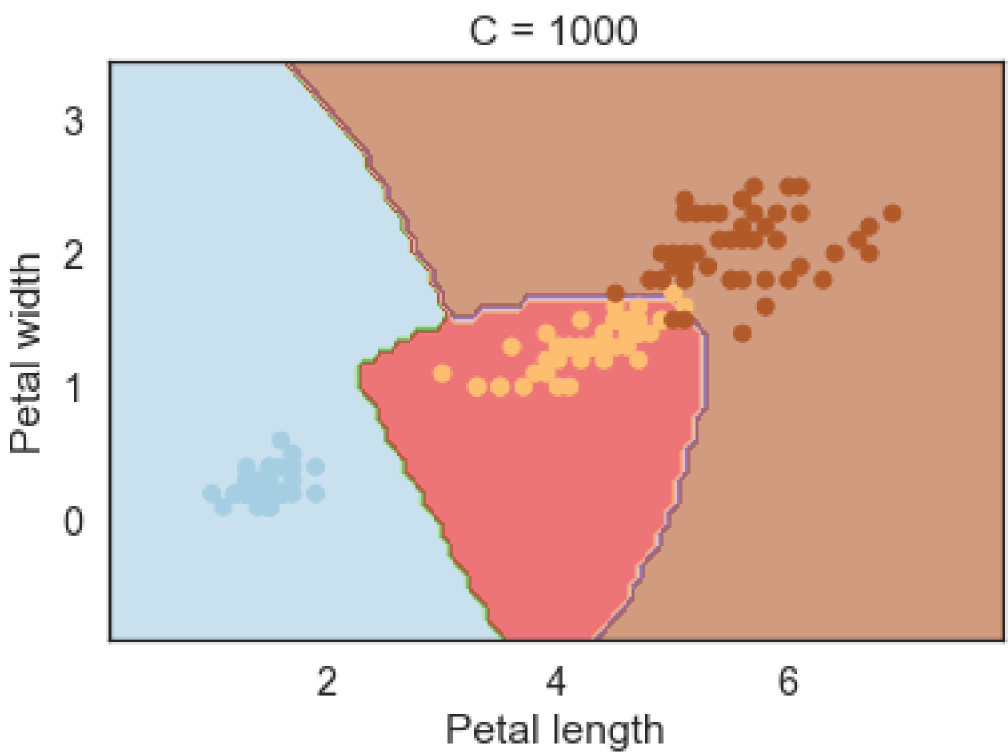



Figure 24:

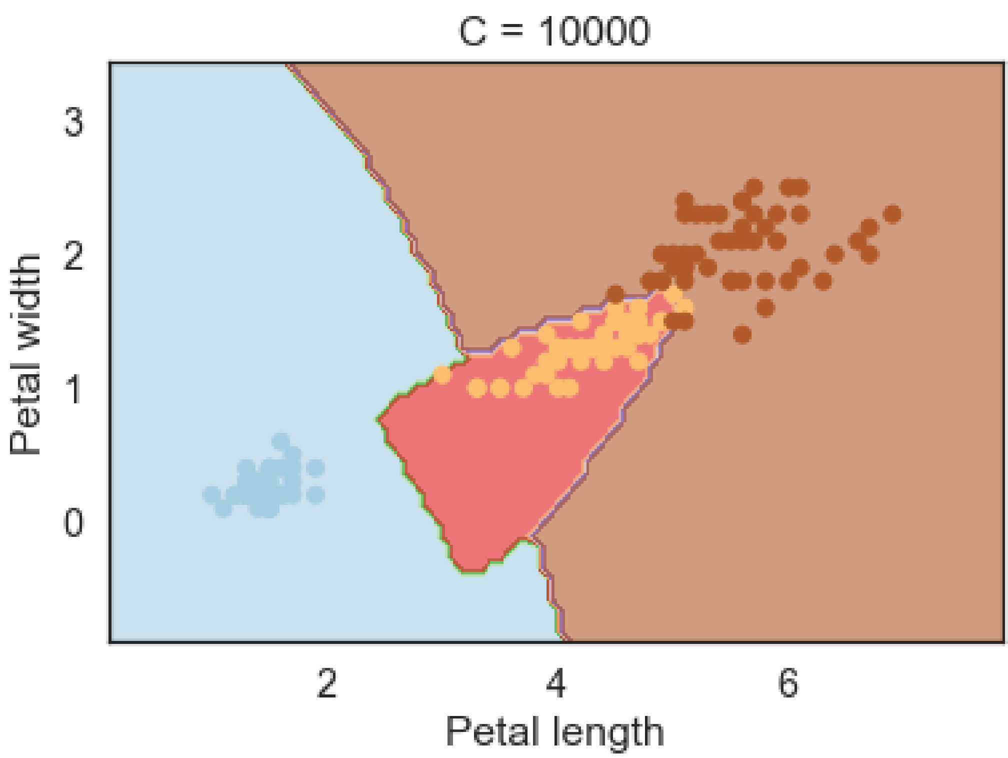



Figure 25:

[51]:
```
degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
xlabel = 'Petal length'
ylabel = 'Petal width'

for degree in degrees:
    svc = svm.SVC(kernel='poly', degree=degree).fit(X, y)
    plotSVC('degree = ' + str(degree), xlabel, ylabel)
```

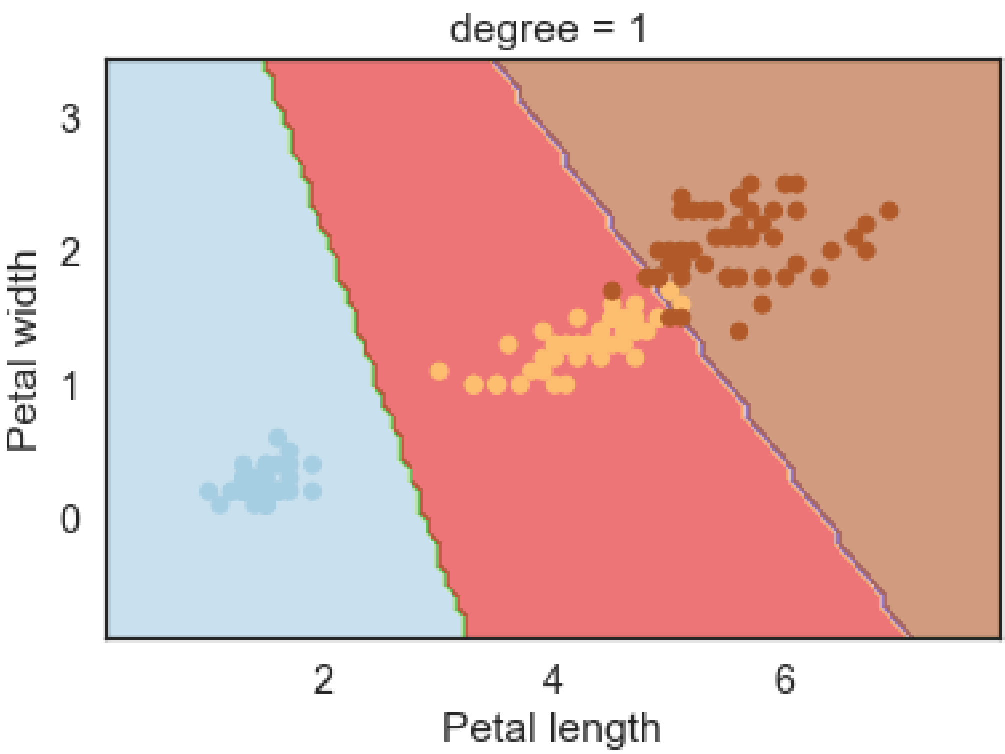



Figure 26:

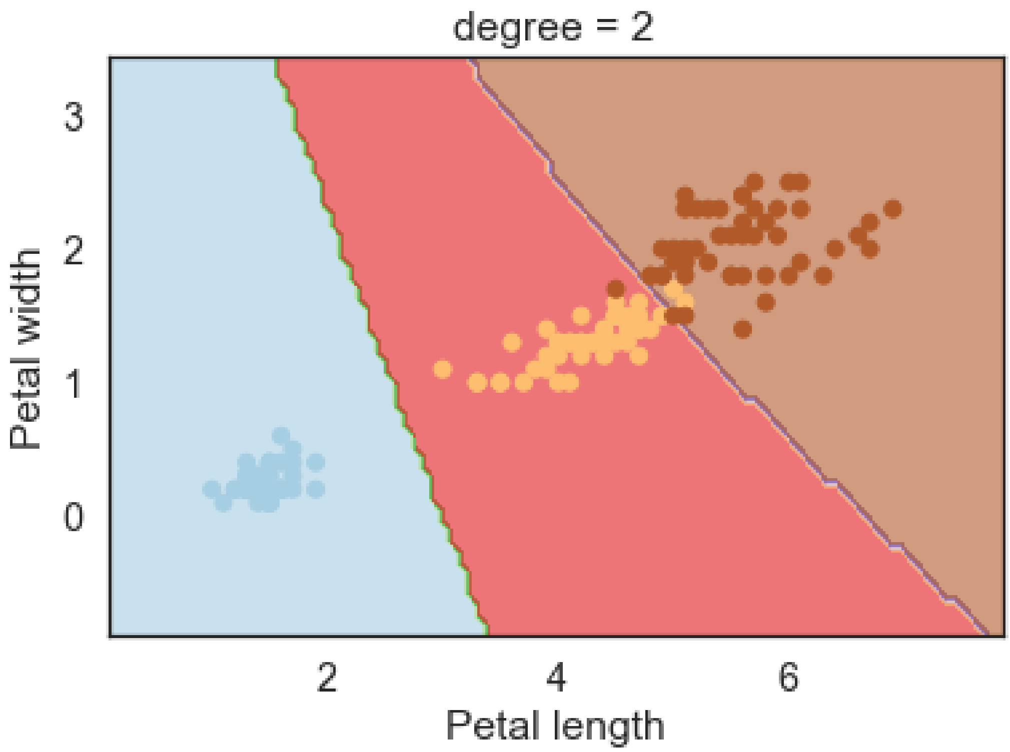



Figure 27:

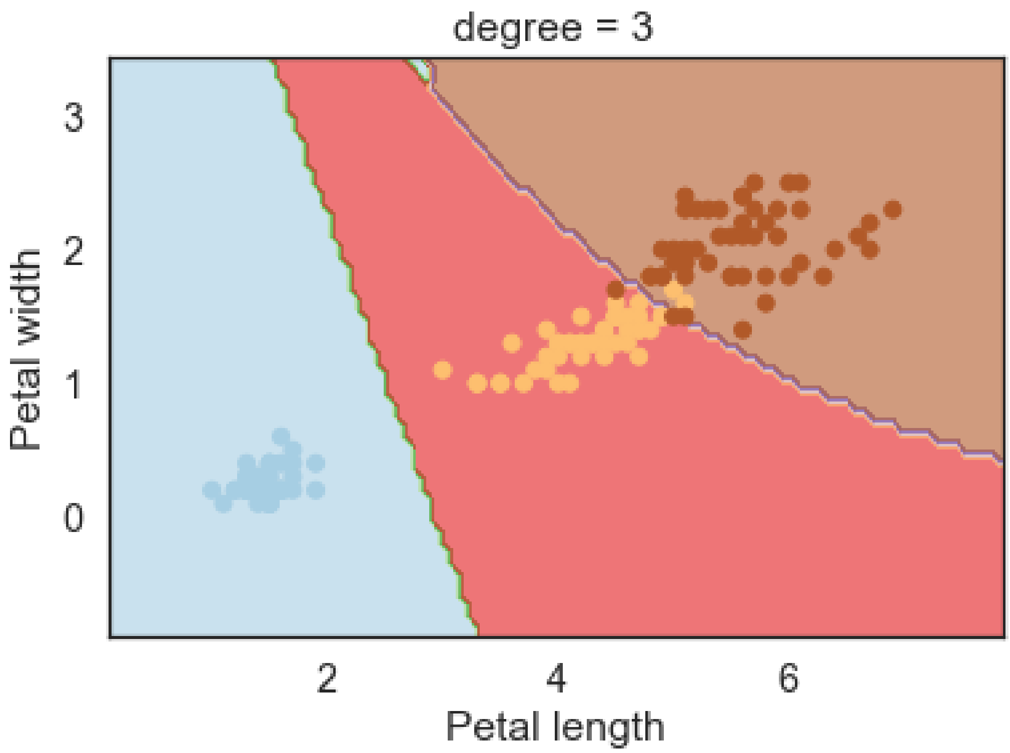



Figure 28:

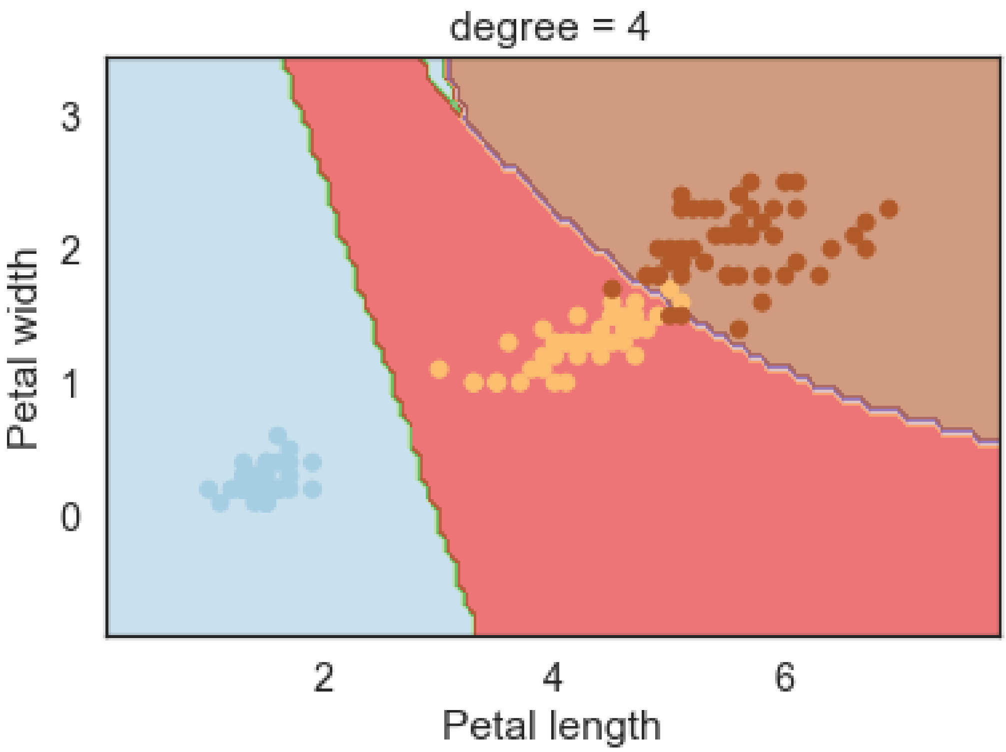



Figure 29:

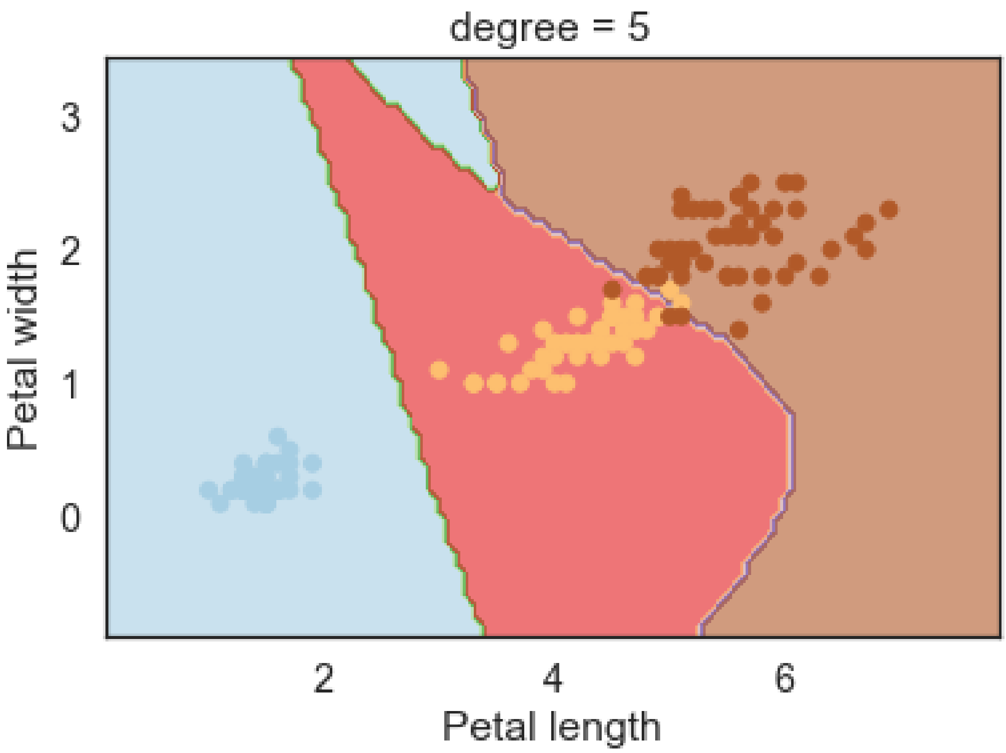



Figure 30:

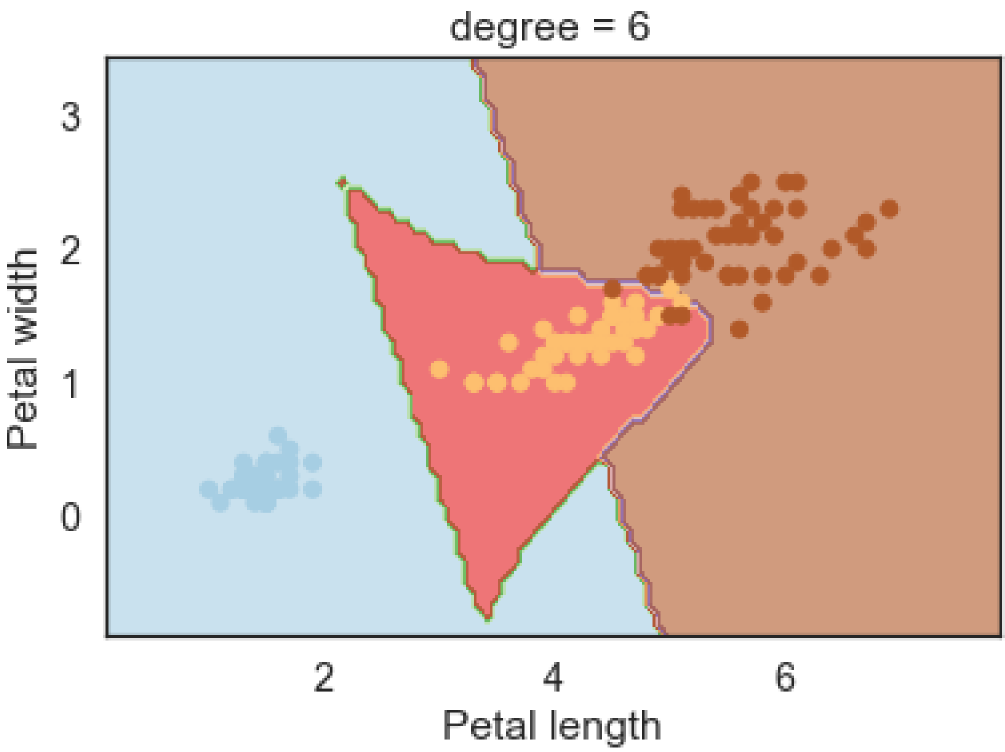



Figure 31:

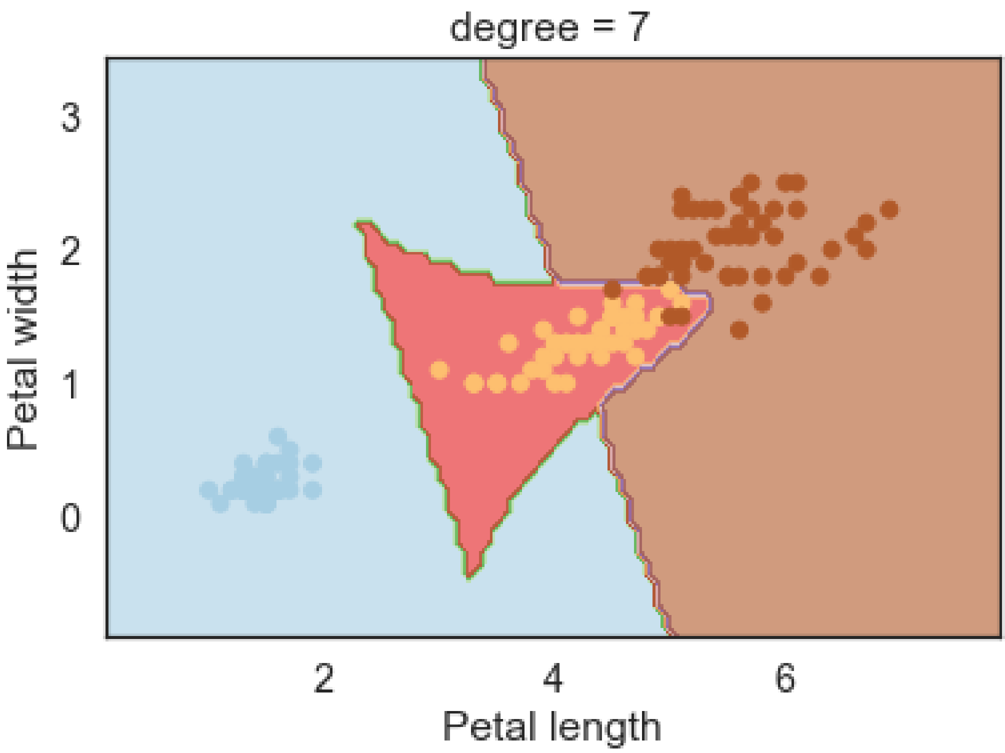



Figure 32:

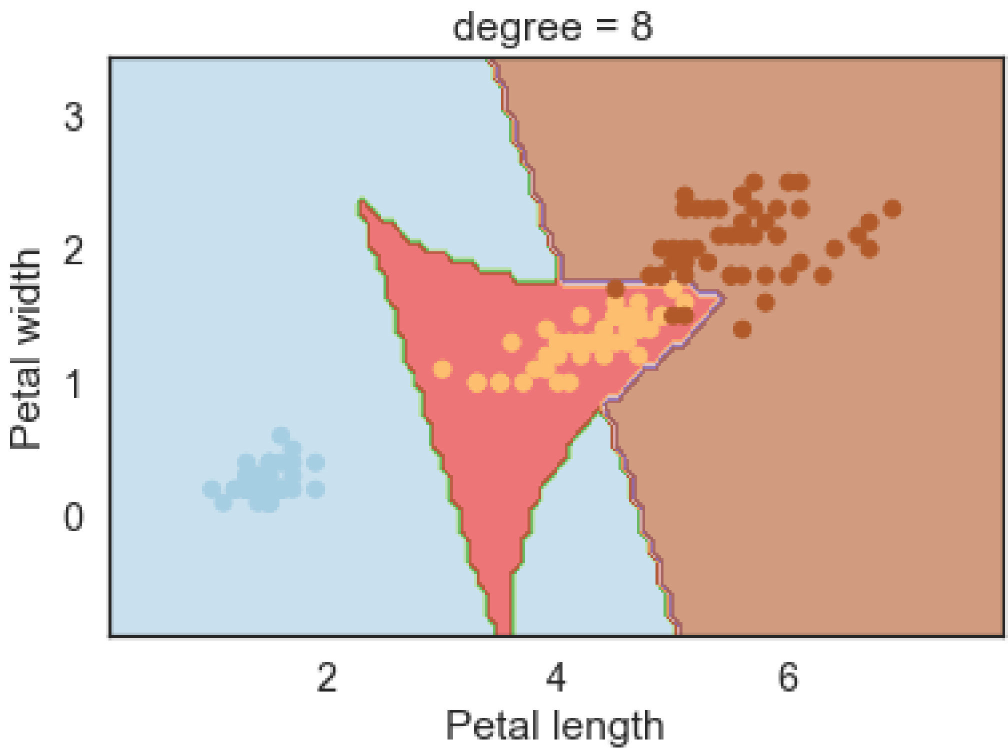



Figure 33:

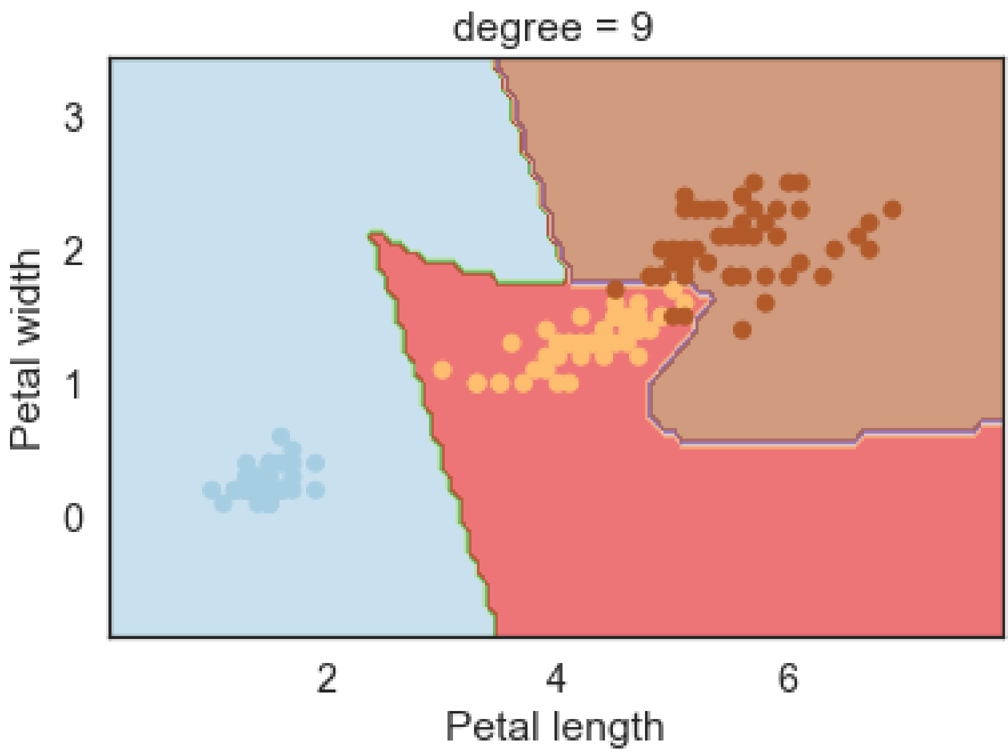



Figure 34:

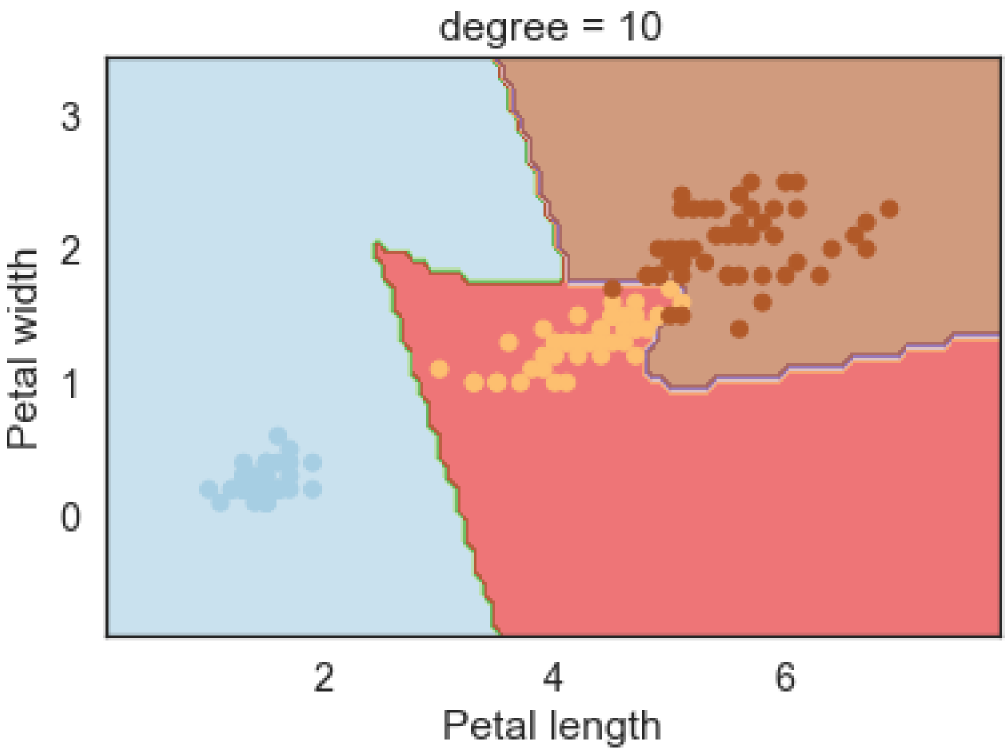



Figure 35: