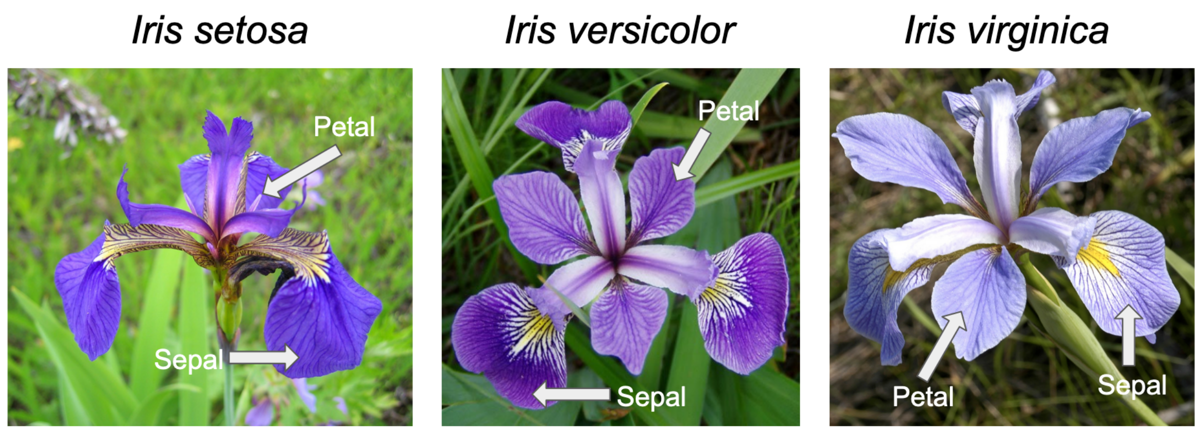
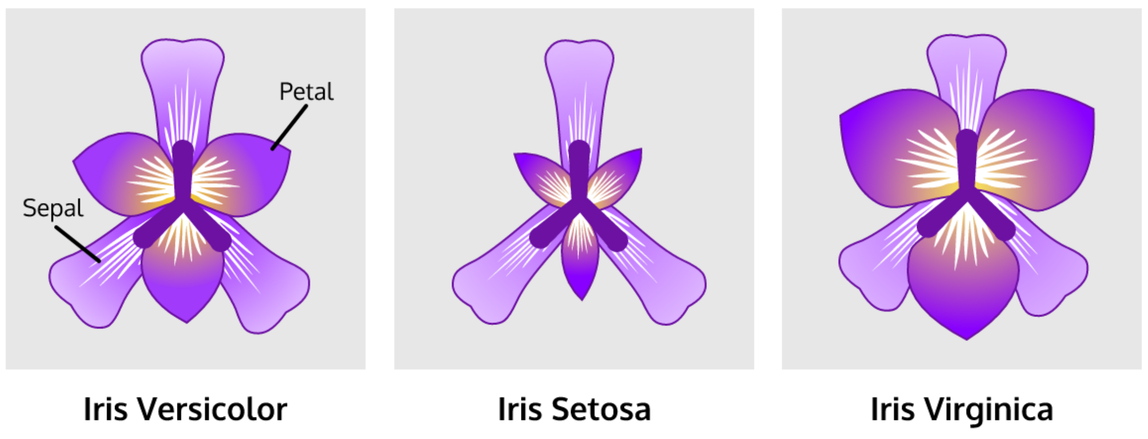
ProjectIris

Storyline: A company wants a model that can predict species of flowers, and asks its engineers to come up with a prototype in three weeks. So can we use Machine learning to predict a flower with species of three?

Introduction

Dataset





The Iris flower dataset or Fisher's Iris data set is a multivariate dataset, introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper. The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis. It is sometimes called Anderson's Iris dataset because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus". Fisher's paper was published in the journal, the Annals of Eugenics, creating controversy about the continued use of the Iris dataset for teaching statistical techniques today.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Goals

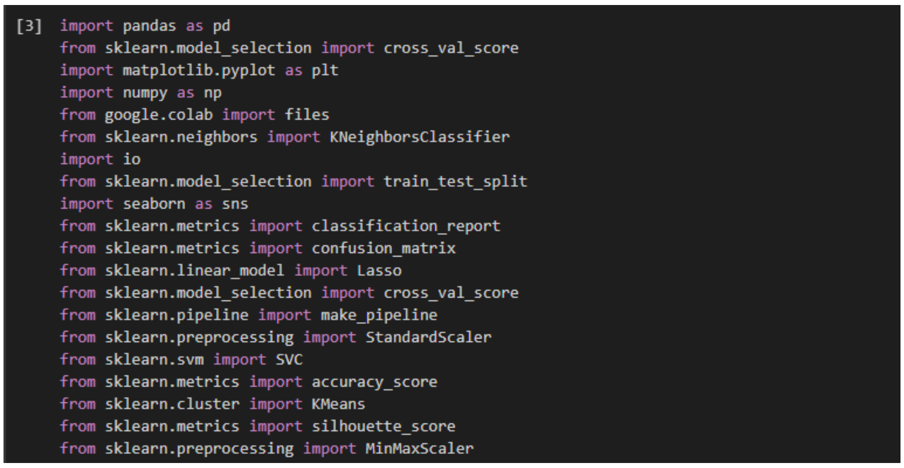
My goals are first, to predict the species of Iris flower by their petal and sepal measurements. In other words, successfully making a functional machine learning model. Second, while doing so, I'd like to explore my chosen dataset, 'Iris Dataset', a rather simple and popular dataset, perfect for beginners. I will use my data analyst skills that I have learned, in hope that they could light me the way to what contained inside this wonderful dataset. And I'd like to use this opportunity to gain more knowledge and sharpen my skills. In other words, my second goal is to understand this dataset, and while doing so, I hope to gain even more experience on data analysis and machine learning.

Tools

Dataset → 'Iris Dataset'

Machine learning model → K-neighbours, SVM, and K-Means Clustering

Used libraries



```
[3]  import pandas as pd
     from sklearn.model_selection import cross_val_score
     import matplotlib.pyplot as plt
     import numpy as np
     from google.colab import files
     from sklearn.neighbors import KNeighborsClassifier
     import io
     from sklearn.model_selection import train_test_split
     import seaborn as sns
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix
     from sklearn.linear_model import Lasso
     from sklearn.model_selection import cross_val_score
     from sklearn.pipeline import make_pipeline
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score
     from sklearn.cluster import KMeans
     from sklearn.metrics import silhouette_score
     from sklearn.preprocessing import MinMaxScaler
```

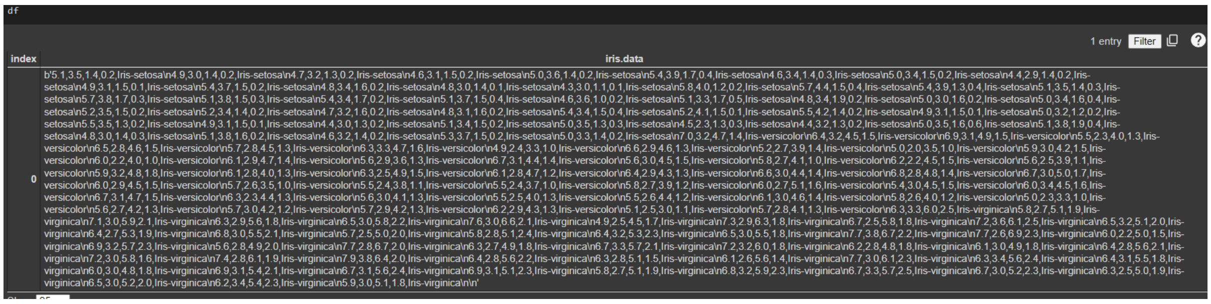
Main event

Stage 1-2: Collecting & Storing

First and second stages aren't too problematic, I simply download the Iris dataset from the UCI Machine Learning Repository website into my computer and store it in a folder on my harddrive. Since it is a small dataset, I had no problem at all.

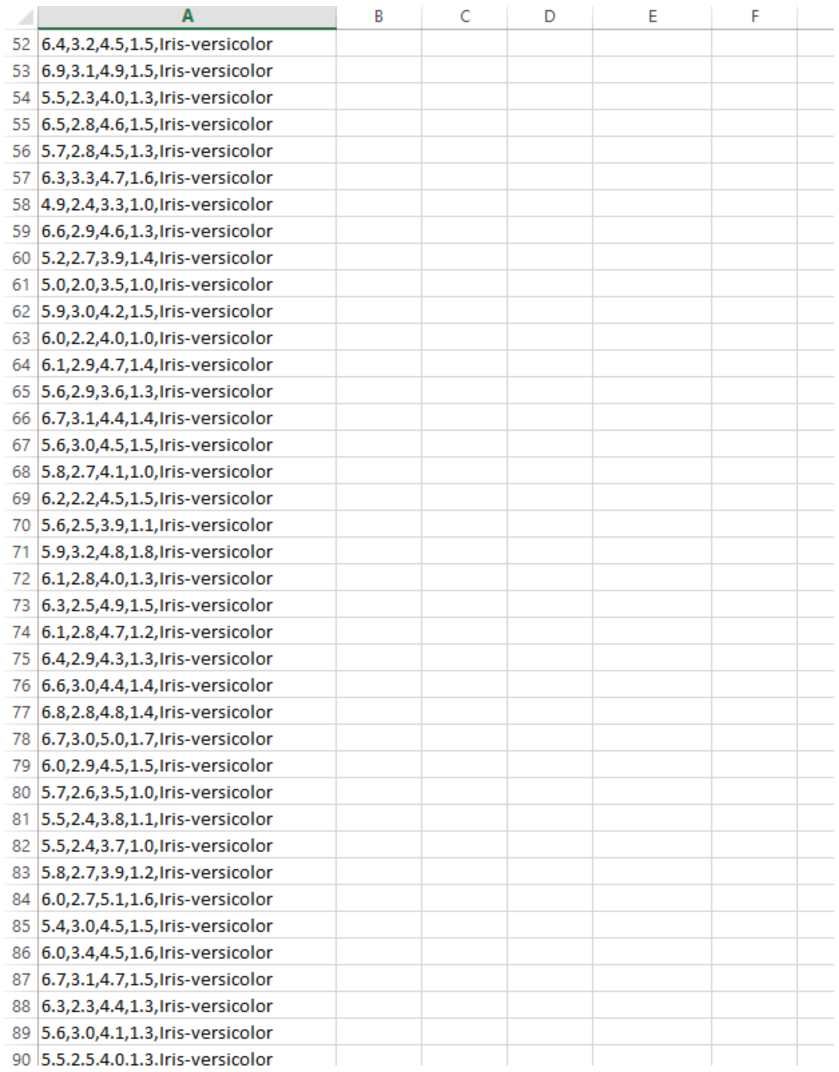
Stage 3: Preparation

Things start to get interesting in this stage, when I examine my dataset, I found out that all the features are grouped together in a single cell. So when I load it up on to my python editor to turn the dataset into DataFrame, everything is on the single row.



```
df
                                                                                         1 entry  Filter
index                                          iris.data
  0   b'5.1,3.5,1.4,0.2,Iris-setosa\n4.9,3.0,1.4,0.2,Iris-setosa\n4.7,3.2,1.3,0.2,Iris-setosa\n4.6,3.1,1.5,0.2,Iris-setosa\n5.0,3.6,1.4,0.2,Iris-setosa\n5.4,3.9,1.7,0.4,Iris-setosa\n4.6,3.4,1.4,0.3,Iris-setosa\n5.0,3.4,1.5,0.2,Iris-setosa\n4.4,2.9,1.4,0.2,Iris-
      setosa\n4.9,3.1,1.5,0.1,Iris-setosa\n5.4,3.7,1.5,0.2,Iris-setosa\n4.8,3.4,1.6,0.2,Iris-setosa\n4.8,3.0,1.4,0.1,Iris-setosa\n4.3,3.0,1.1,0.1,Iris-setosa\n5.8,4.0,1.2,0.2,Iris-setosa\n5.7,4.4,1.5,0.4,Iris-setosa\n5.4,3.9,1.3,0.4,Iris-setosa\n5.1,3.5,1.4,0.3,Iris-
      setosa\n5.7,3.8,1.7,0.3,Iris-setosa\n5.1,3.8,1.5,0.3,Iris-setosa\n5.4,3.4,1.7,0.2,Iris-setosa\n5.1,3.7,1.5,0.4,Iris-setosa\n4.6,3.6,1.0,0.2,Iris-setosa\n5.1,3.3,1.7,0.5,Iris-setosa\n4.8,3.4,1.9,0.2,Iris-setosa\n5.0,3.0,1.6,0.2,Iris-setosa\n5.0,3.4,1.6,0.4,Iris-
      setosa\n5.2,3.5,1.5,0.2,Iris-setosa\n5.2,3.4,1.4,0.2,Iris-setosa\n4.7,3.2,1.6,0.2,Iris-setosa\n4.8,3.1,1.6,0.2,Iris-setosa\n5.4,3.4,1.5,0.4,Iris-setosa\n5.2,4.1,1.5,0.1,Iris-setosa\n5.5,4.2,1.4,0.2,Iris-setosa\n4.9,3.1,1.5,0.1,Iris-setosa\n5.0,3.2,1.2,0.2,Iris-
      setosa\n5.5,3.5,1.3,0.2,Iris-setosa\n4.9,3.1,1.5,0.1,Iris-setosa\n4.4,3.0,1.3,0.2,Iris-setosa\n5.1,3.4,1.5,0.2,Iris-setosa\n5.0,3.5,1.3,0.3,Iris-setosa\n4.5,2.3,1.3,0.3,Iris-setosa\n4.4,3.2,1.3,0.2,Iris-setosa\n5.0,3.5,1.6,0.6,Iris-setosa\n5.1,3.8,1.9,0.4,Iris-
      setosa\n4.8,3.0,1.4,0.3,Iris-setosa\n5.1,3.8,1.6,0.2,Iris-setosa\n4.6,3.2,1.4,0.2,Iris-setosa\n5.3,3.7,1.5,0.2,Iris-setosa\n5.0,3.3,1.4,0.2,Iris-setosa\n7.0,3.2,4.7,1.4,Iris-versicolor\n6.4,3.2,4.5,1.5,Iris-versicolor\n6.9,3.1,4.9,1.5,Iris-versicolor\n5.5,2.3,4.0,1.3,Iris-
      versicolor\n6.5,2.8,4.6,1.5,Iris-versicolor\n5.7,2.8,4.5,1.3,Iris-versicolor\n6.3,3.3,4.7,1.6,Iris-versicolor\n4.9,2.4,3.3,1.0,Iris-versicolor\n6.6,2.9,4.6,1.3,Iris-versicolor\n5.2,2.7,3.9,1.4,Iris-versicolor\n5.0,2.0,3.5,1.0,Iris-versicolor\n5.9,3.0,4.2,1.5,Iris-
      versicolor\n6.0,2.2,4.0,1.0,Iris-versicolor\n6.1,2.9,4.7,1.4,Iris-versicolor\n5.6,2.9,3.6,1.3,Iris-versicolor\n6.7,3.1,4.4,1.4,Iris-versicolor\n5.6,3.0,4.5,1.5,Iris-versicolor\n5.8,2.7,4.1,1.0,Iris-versicolor\n6.2,2.2,4.5,1.5,Iris-versicolor\n5.6,2.5,3.9,1.1,Iris-
      versicolor\n5.9,3.2,4.8,1.8,Iris-versicolor\n6.1,2.8,4.0,1.3,Iris-versicolor\n6.3,2.5,4.9,1.5,Iris-versicolor\n6.1,2.8,4.7,1.2,Iris-versicolor\n6.4,2.9,4.3,1.3,Iris-versicolor\n6.6,3.0,4.4,1.4,Iris-versicolor\n6.8,2.8,4.8,1.4,Iris-versicolor\n6.7,3.0,5.0,1.7,Iris-
      versicolor\n6.0,2.9,4.5,1.5,Iris-versicolor\n5.7,2.6,3.5,1.0,Iris-versicolor\n5.5,2.4,3.8,1.1,Iris-versicolor\n5.5,2.4,3.7,1.0,Iris-versicolor\n5.8,2.7,3.9,1.2,Iris-versicolor\n6.0,2.7,5.1,1.6,Iris-versicolor\n5.4,3.0,4.5,1.5,Iris-versicolor\n6.0,3.4,4.5,1.6,Iris-
      versicolor\n6.7,3.1,4.7,1.5,Iris-versicolor\n6.3,2.3,4.4,1.3,Iris-versicolor\n5.6,3.0,4.1,1.3,Iris-versicolor\n5.5,2.5,4.0,1.3,Iris-versicolor\n5.5,2.6,4.4,1.2,Iris-versicolor\n6.1,3.0,4.6,1.4,Iris-versicolor\n5.8,2.6,4.0,1.2,Iris-versicolor\n5.0,2.3,3.3,1.0,Iris-
      versicolor\n5.6,2.7,4.2,1.3,Iris-versicolor\n5.7,3.0,4.2,1.2,Iris-versicolor\n5.7,2.9,4.2,1.3,Iris-versicolor\n6.2,2.9,4.3,1.3,Iris-versicolor\n5.1,2.5,3.0,1.1,Iris-versicolor\n5.7,2.8,4.1,1.3,Iris-versicolor\n6.3,3.3,6.0,2.5,Iris-virginica\n5.8,2.7,5.1,1.9,Iris-
      virginica\n7.1,3.0,5.9,2.1,Iris-virginica\n6.3,2.9,5.6,1.8,Iris-virginica\n6.5,3.0,5.8,2.2,Iris-virginica\n7.6,3.0,6.6,2.1,Iris-virginica\n4.9,2.5,4.5,1.7,Iris-virginica\n7.3,2.9,6.3,1.8,Iris-virginica\n6.7,2.5,5.8,1.8,Iris-virginica\n7.2,3.6,6.1,2.5,Iris-virginica\n6.5,3.2,5.1,2.0,Iris-
      virginica\n6.4,2.7,5.3,1.9,Iris-virginica\n6.8,3.0,5.5,2.1,Iris-virginica\n5.7,2.5,5.0,2.0,Iris-virginica\n5.8,2.8,5.1,2.4,Iris-virginica\n6.4,3.2,5.3,2.3,Iris-virginica\n6.5,3.0,5.5,1.8,Iris-virginica\n7.7,3.8,6.7,2.2,Iris-virginica\n7.7,2.6,6.9,2.3,Iris-virginica\n6.0,2.2,5.0,1.5,Iris-
      virginica\n6.9,3.2,5.7,2.3,Iris-virginica\n5.6,2.8,4.9,2.0,Iris-virginica\n7.7,2.8,6.7,2.0,Iris-virginica\n6.3,2.7,4.9,1.8,Iris-virginica\n6.7,3.3,5.7,2.1,Iris-virginica\n7.2,3.2,6.0,1.8,Iris-virginica\n6.2,2.8,4.8,1.8,Iris-virginica\n6.1,3.0,4.9,1.8,Iris-virginica\n6.4,2.8,5.6,2.1,Iris-
      virginica\n7.2,3.0,5.8,1.6,Iris-virginica\n7.4,2.8,6.1,1.9,Iris-virginica\n7.9,3.8,6.4,2.0,Iris-virginica\n6.4,2.8,5.6,2.2,Iris-virginica\n6.3,2.8,5.1,1.5,Iris-virginica\n6.1,2.6,5.6,1.4,Iris-virginica\n7.7,3.0,6.1,2.3,Iris-virginica\n6.3,3.4,5.6,2.4,Iris-virginica\n6.4,3.1,5.5,1.8,Iris-
      virginica\n6.0,3.0,4.8,1.8,Iris-virginica\n6.9,3.1,5.4,2.1,Iris-virginica\n6.7,3.1,5.6,2.4,Iris-virginica\n6.9,3.1,5.1,2.3,Iris-virginica\n5.8,2.7,5.1,1.9,Iris-virginica\n6.8,3.2,5.9,2.3,Iris-virginica\n6.7,3.3,5.7,2.5,Iris-virginica\n6.7,3.0,5.2,2.3,Iris-virginica\n6.3,2.5,5.0,1.9,Iris-
      virginica\n6.5,3.0,5.2,2.0,Iris-virginica\n6.2,3.4,5.4,2.3,Iris-virginica\n5.9,3.0,5.1,1.8,Iris-virginica\n\n'
```

This is where preparation stage started for my project. My method of fixing this is to open the dataset on Excel.



	A	B	C	D	E	F
52	6.4,3.2,4.5,1.5,Iris-versicolor					
53	6.9,3.1,4.9,1.5,Iris-versicolor					
54	5.5,2.3,4.0,1.3,Iris-versicolor					
55	6.5,2.8,4.6,1.5,Iris-versicolor					
56	5.7,2.8,4.5,1.3,Iris-versicolor					
57	6.3,3.3,4.7,1.6,Iris-versicolor					
58	4.9,2.4,3.3,1.0,Iris-versicolor					
59	6.6,2.9,4.6,1.3,Iris-versicolor					
60	5.2,2.7,3.9,1.4,Iris-versicolor					
61	5.0,2.0,3.5,1.0,Iris-versicolor					
62	5.9,3.0,4.2,1.5,Iris-versicolor					
63	6.0,2.2,4.0,1.0,Iris-versicolor					
64	6.1,2.9,4.7,1.4,Iris-versicolor					
65	5.6,2.9,3.6,1.3,Iris-versicolor					
66	6.7,3.1,4.4,1.4,Iris-versicolor					
67	5.6,3.0,4.5,1.5,Iris-versicolor					
68	5.8,2.7,4.1,1.0,Iris-versicolor					
69	6.2,2.2,4.5,1.5,Iris-versicolor					
70	5.6,2.5,3.9,1.1,Iris-versicolor					
71	5.9,3.2,4.8,1.8,Iris-versicolor					
72	6.1,2.8,4.0,1.3,Iris-versicolor					
73	6.3,2.5,4.9,1.5,Iris-versicolor					
74	6.1,2.8,4.7,1.2,Iris-versicolor					
75	6.4,2.9,4.3,1.3,Iris-versicolor					
76	6.6,3.0,4.4,1.4,Iris-versicolor					
77	6.8,2.8,4.8,1.4,Iris-versicolor					
78	6.7,3.0,5.0,1.7,Iris-versicolor					
79	6.0,2.9,4.5,1.5,Iris-versicolor					
80	5.7,2.6,3.5,1.0,Iris-versicolor					
81	5.5,2.4,3.8,1.1,Iris-versicolor					
82	5.5,2.4,3.7,1.0,Iris-versicolor					
83	5.8,2.7,3.9,1.2,Iris-versicolor					
84	6.0,2.7,5.1,1.6,Iris-versicolor					
85	5.4,3.0,4.5,1.5,Iris-versicolor					
86	6.0,3.4,4.5,1.6,Iris-versicolor					
87	6.7,3.1,4.7,1.5,Iris-versicolor					
88	6.3,2.3,4.4,1.3,Iris-versicolor					
89	5.6,3.0,4.1,1.3,Iris-versicolor					
90	5.5,2.5,4.0,1.3,Iris-versicolor					

And explode each value into their own cells by using the MID function. Also giving each feature a name while doing so.

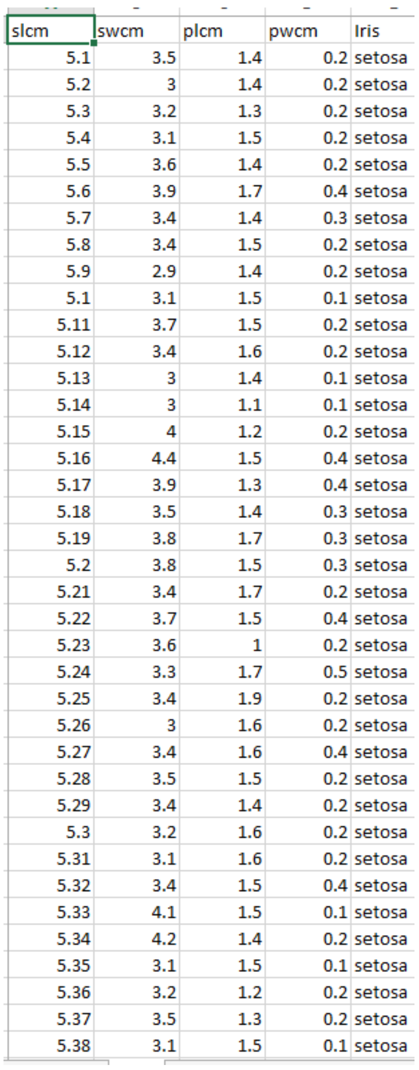
‘slcm’ → sepal length in centimeter

‘swcm’ → sepal width in centimeter

‘plcm’ → petal length in centimeter

‘pwcm’ → petal width in centimeter

,and ‘Iris’ is just variable name for species



slcm	swcm	plcm	pwcm	Iris
5.1	3.5	1.4	0.2	setosa
5.2	3	1.4	0.2	setosa
5.3	3.2	1.3	0.2	setosa
5.4	3.1	1.5	0.2	setosa
5.5	3.6	1.4	0.2	setosa
5.6	3.9	1.7	0.4	setosa
5.7	3.4	1.4	0.3	setosa
5.8	3.4	1.5	0.2	setosa
5.9	2.9	1.4	0.2	setosa
5.1	3.1	1.5	0.1	setosa
5.11	3.7	1.5	0.2	setosa
5.12	3.4	1.6	0.2	setosa
5.13	3	1.4	0.1	setosa
5.14	3	1.1	0.1	setosa
5.15	4	1.2	0.2	setosa
5.16	4.4	1.5	0.4	setosa
5.17	3.9	1.3	0.4	setosa
5.18	3.5	1.4	0.3	setosa
5.19	3.8	1.7	0.3	setosa
5.2	3.8	1.5	0.3	setosa
5.21	3.4	1.7	0.2	setosa
5.22	3.7	1.5	0.4	setosa
5.23	3.6	1	0.2	setosa
5.24	3.3	1.7	0.5	setosa
5.25	3.4	1.9	0.2	setosa
5.26	3	1.6	0.2	setosa
5.27	3.4	1.6	0.4	setosa
5.28	3.5	1.5	0.2	setosa
5.29	3.4	1.4	0.2	setosa
5.3	3.2	1.6	0.2	setosa
5.31	3.1	1.6	0.2	setosa
5.32	3.4	1.5	0.4	setosa
5.33	4.1	1.5	0.1	setosa
5.34	4.2	1.4	0.2	setosa
5.35	3.1	1.5	0.1	setosa
5.36	3.2	1.2	0.2	setosa
5.37	3.5	1.3	0.2	setosa
5.38	3.1	1.5	0.1	setosa

There are no missing or outlier values. Therefore, the preparation stage for my dataset has concluded. We will dig into this pattern more in the next section where Exploratory of Data analysis and data visualisation would be applied to full effect to understand pattern(s) in this dataset.

Stage 4: EDA & Visualisation

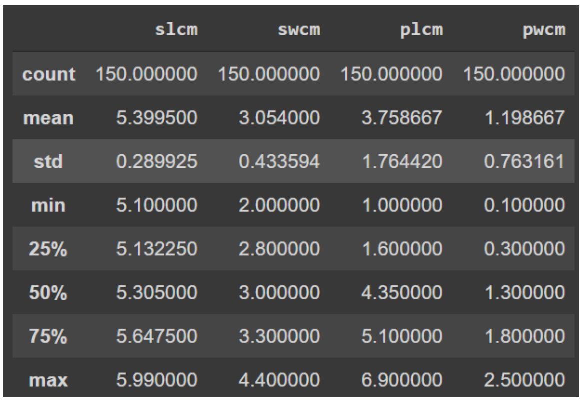
What is data visualisation and why is it important?

Data visualisation is the representation of data or information in a graph, chart, or other visual format. It communicates relationships of the data with images. This is important because it allows trends and patterns to be more easily seen. We need data visualisation because a visual summary of information makes it easier to identify patterns and trends than looking through thousands of rows on a spreadsheet.

Exploratory data analysis

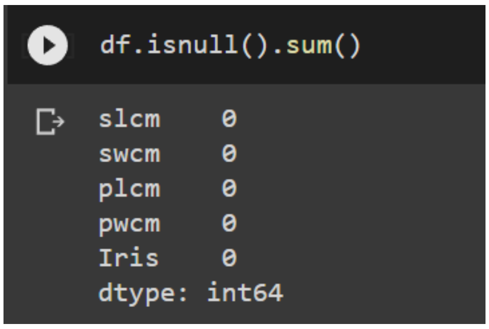
EDA is used by data scientists to analyse and investigate data sets and summarise their main characteristics, often employing data visualisation methods.

This section is going to be about studying the dataset, once all the data patterns are understood, it is the goal to find the essential features that we could fit into our Machine learning model.



	slcm	swcm	plcm	pwcm
count	150.000000	150.000000	150.000000	150.000000
mean	5.399500	3.054000	3.758667	1.198667
std	0.289925	0.433594	1.764420	0.763161
min	5.100000	2.000000	1.000000	0.100000
25%	5.132250	2.800000	1.600000	0.300000
50%	5.305000	3.000000	4.350000	1.300000
75%	5.647500	3.300000	5.100000	1.800000
max	5.990000	4.400000	6.900000	2.500000

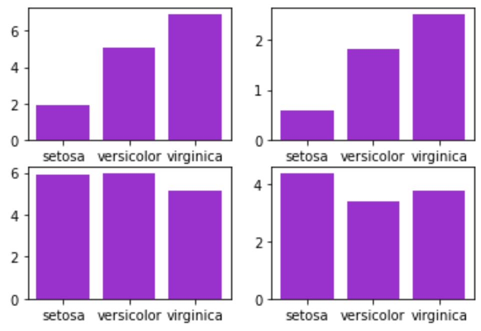
Here is a table that contains, count, mean, standard deviation, min, 25th, 50th, and 75th percentile, and max for each feature.



```
df.isnull().sum()

slcm     0
swcm     0
plcm     0
pwcm     0
Iris     0
dtype: int64
```

To be certain, I have checked for any missing values. There is nothing missing.

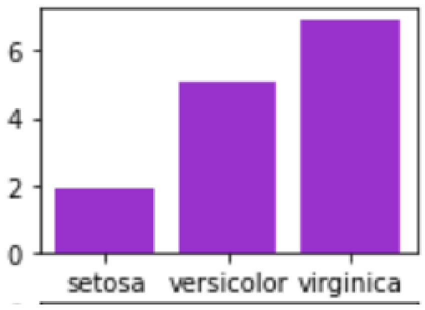
Now to the main event, while I was “Cleaning” my dataset in the previous section. I’ve noticed a pattern in my dataset. The pattern can be seen when you examine the petal length, petal width features. These species each have very unique petal sizes when compared to one another. Starting from ‘Setosa’ species, which has the smallest petal size out of the bunch, then ‘Versicolor’ in the middle, and lastly, the biggest out of them all, ‘virginica.



Subplots 1

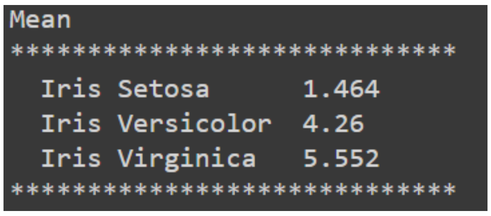
```
fig, ax = plt.subplots(2,2)
ax[0, 0].bar(df['Iris'], df['plcm'], color = 'darkorchid')
ax[0, 1].bar(df['Iris'], df['pwcm'], color = 'darkorchid')
ax[1, 0].bar(df['Iris'], df['slcm'], color = 'darkorchid')
ax[1, 1].bar(df['Iris'], df['swcm'], color = 'darkorchid')
plt.show()
```

I have added the lines of code, so we can visualise and understand the bar graphs better. As you can see, the first graph on the top left is ‘plcm’ or petal length in centimeter.



It is obvious that there are differences in sizes across species. First, Setosa comes in just less than or equal to 2 centimeters. Second, Versicolor, the length has doubled when compared to Setosa’s petal length, about 4.5 centimeters long, and no less than 3 centimeters long. Lastly, Virginica, this time, the length hasn’t been doubled when compared to previous species, however there is still a significant change in size. Virginica’s petal length comes in at approximately 5.5 centimeters. A matter of course, these figures mentioned were retrieved from the bar graph above.

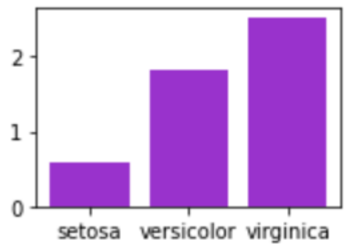
```
iris_setosa = df[df['Iris'] == 'setosa']['plcm']
iris_versicolor = df[df['Iris'] == 'versicolor']['plcm']
iris_virginica = df[df['Iris'] == 'virginica']['plcm']
print('Mean')
print('*****************************')
print('  Iris Setosa\t  ', np.mean(iris_setosa))
print('  Iris Versicolor ', np.mean(iris_versicolor))
print('  Iris Virginica  ',  np.mean(iris_virginica))
print('*****************************',end = '\n\n\n')
```



```
Mean
*****************************
  Iris Setosa      1.464
  Iris Versicolor  4.26
  Iris Virginica   5.552
*****************************
```

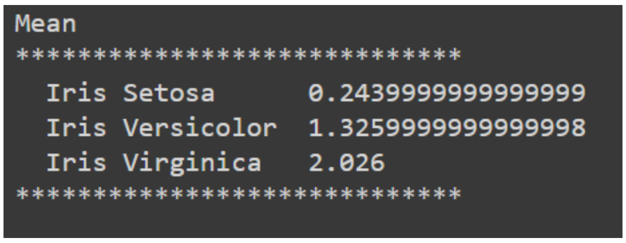
The figures were generated by these lines of code

Moving on to the next graph,



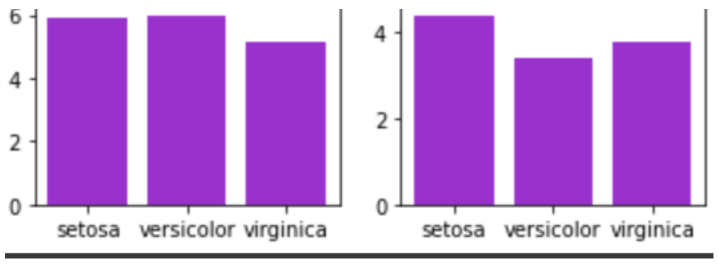
Moving on to the next graph, This is one of the graphs from ‘Subplot 1’, this is ‘pwcm’ or petal width in centimeters. As you can, the difference in width across species is quite obvious. Again, Setosa is the smallest out of the bunch, with its mean only 0.2 centimeter in width. Second, Versicolor with its mean width to be around 1.3 centimeters. And, lastly, virginica with just about 2 centimeters.

```
iris_setosa_1 = df[df['Iris'] == 'setosa']['pwcm']
iris_versicolor_1 = df[df['Iris'] == 'versicolor']['pwcm']
iris_virginica_1 = df[df['Iris'] == 'virginica']['pwcm']

print('Mean')
print('*****************************')
print('  Iris Setosa\t  ', np.mean(iris_setosa_1))
print('  Iris Versicolor ', np.mean(iris_versicolor_1))
print('  Iris Virginica  ',  np.mean(iris_virginica_1))
print('*****************************',end = '\n\n\n')
```



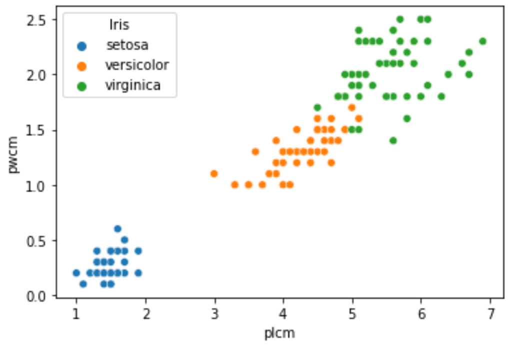
```
Mean
*****************************
  Iris Setosa      0.2439999999999999
  Iris Versicolor  1.3259999999999998
  Iris Virginica   2.026
*****************************
```

This pattern shows that Setosa has the smallest measurements and Virginica has the biggest measurements, and how significant these species inequality compared to one another is an encouraging sign. We could use these two features to classify the flowers.



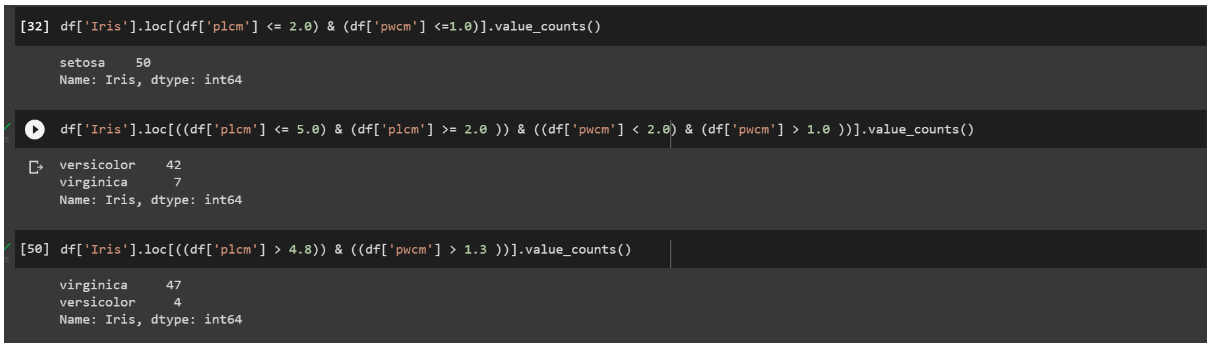
Before moving on, let's have a look at our last two graphs. The one on the left is 'slcm' or 'sepal' length in centimeters. As you can see, There isn't really anything going on here. If I were to ask someone to distinguish these irises by using 'slcm' as the main metric. Most likely they wouldn't be able to succeed because these species are so similar in sepal sizes. We could say the same thing with the graph on the right, 'swcm' or sepal width in centimeters.

To make it even clearer,



```
sns.scatterplot(x='plcm', y='pwcm', data=df, hue='Iris')
```

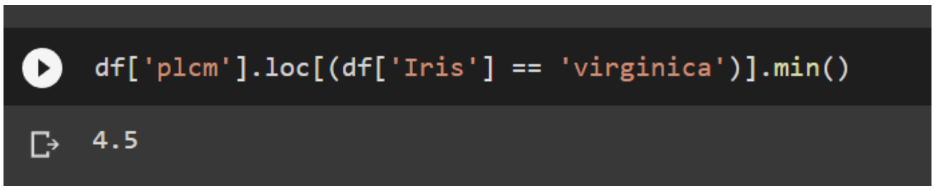
What you are seeing is a scatterplot of 'plcm' as x-axis and 'pwcm' as y-axis. The purpose of this plot is to shine lights on patterns hidden within 'plcm' and 'pwcm'. Here, it is now very clear, you could cluster each species very easily, Setosa falls on the left of the graph because of its small size. Versicolor and Virginica are slightly overlapped however they are still distinguishable from each other.



```
[32] df['Iris'].loc[(df['plcm'] <= 2.0) & (df['pwcm'] <=1.0)].value_counts()

     setosa    50
     Name: Iris, dtype: int64

     df['Iris'].loc[((df['plcm'] <= 5.0) & (df['plcm'] >= 2.0 )) & ((df['pwcm'] < 2.0) & (df['pwcm'] > 1.0 ))].value_counts()

     versicolor    42
     virginica      7
     Name: Iris, dtype: int64

[50] df['Iris'].loc[((df['plcm'] > 4.8)) & ((df['pwcm'] > 1.3 ))].value_counts()

     virginica     47
     versicolor     4
     Name: Iris, dtype: int64
```

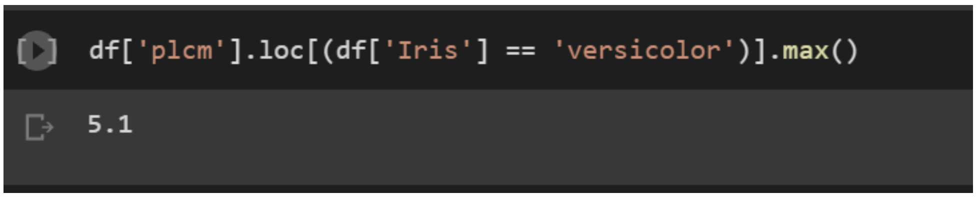
```
if PL <= 2 AND PW <= 1 THEN flower is Iris Setosa
if (PL < 5 AND PL > 2.5) AND (PW < 2 AND PW > 1) THEN flower is Iris Vericolor.
if (PL > 4.8) AND (PW > 1.3) THEN flower is Iris Virginica
```

Here is just me using the figures calculated beforehand of each feature(plcm and pwcm) to see if we could find the target species manually. Setosa is perfect, there is some overlap with Versicolor and Virginica, however it is minimal, which is acceptable.



```
df['plcm'].loc[(df['Iris'] == 'virginica')].min()
```

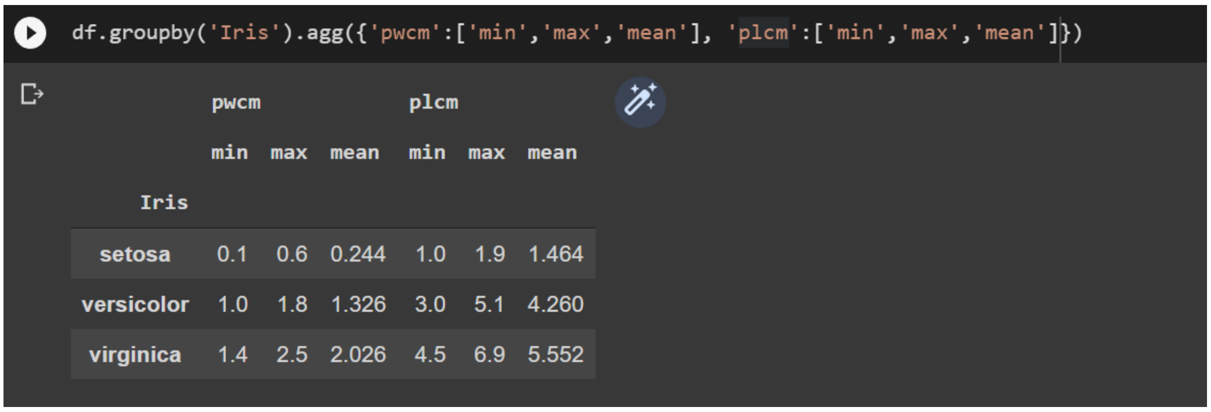
```
4.5
```



```
df['plcm'].loc[(df['Iris'] == 'versicolor')].max()
```

```
5.1
```

Slight overlap with versicolor and virginica



```
df.groupby('Iris').agg({'pwcm':['min','max','mean'], 'plcm':['min','max','mean']})
```

	pwcm			plcm		
	min	max	mean	min	max	mean
Iris						
setosa	0.1	0.6	0.244	1.0	1.9	1.464
versicolor	1.0	1.8	1.326	3.0	5.1	4.260
virginica	1.4	2.5	2.026	4.5	6.9	5.552

Observations

- Petal length and petal width are most useful features to identify specie of the flowers.
- Iris Setosa is linearly separable whereas Iris Versicolor and Iris Virginica have some overlap (almost linearly separable)
- We can easily classify flowers on 2D with minimal errors

Stage 5: Machine learning Model & Visualisation

Intro to ML

Slight intro on what is machine learning, machine learning is a subfield of artificial intelligence, which is broadly defined as **the capability of a machine to imitate intelligent human behavior**. Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.

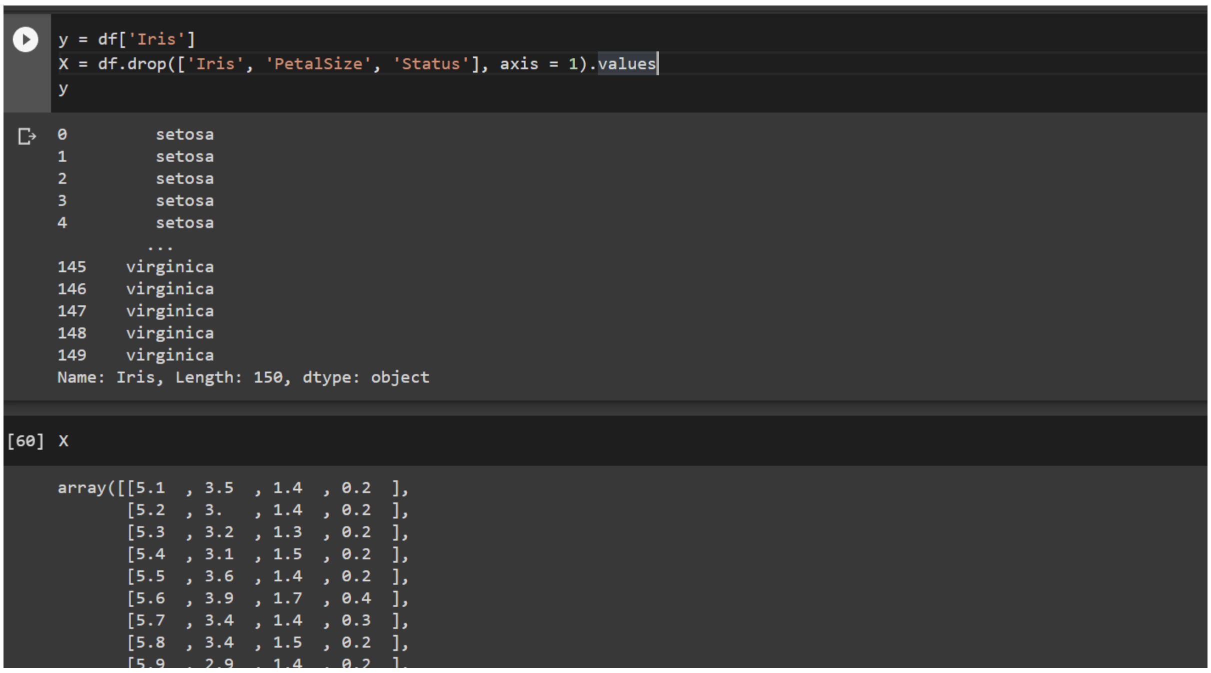
From the previous section, one of our observations is that we could easily classify flowers with a 2D model, with minimal errors. Our dataset is four-dimensional (4 features) which is quite small. In bigger datasets, they might have up to thousand features or thousand-dimensional. Therefore, if I were to train a model with 4D data it would still be acceptable. But I think it is a great practice to always look for essential features and try to reduce our data dimension, for performance sake. However, it is also acceptable to have a little more complex model, if we could achieve more accuracy in return.

Therefore, for this section, I have decided to do both and maybe more...

Let’s get started.

Training a ML model with 4D data

We are going to have a look at the most complex model of this project first, ML model trained with 4D data. I will be using the K-nearest neighbour algorithm. K-nearest neighbors (kNN) is a supervised machine learning algorithm that can be used to solve both classification and regression tasks. We could see KNN as an algorithm that comes from real life. People tend to be affected by the people around them. Our behaviour is guided by the friends we grew up with.



```
y = df['Iris']
X = df.drop(['Iris', 'PetalSize', 'Status'], axis = 1).values
y
```

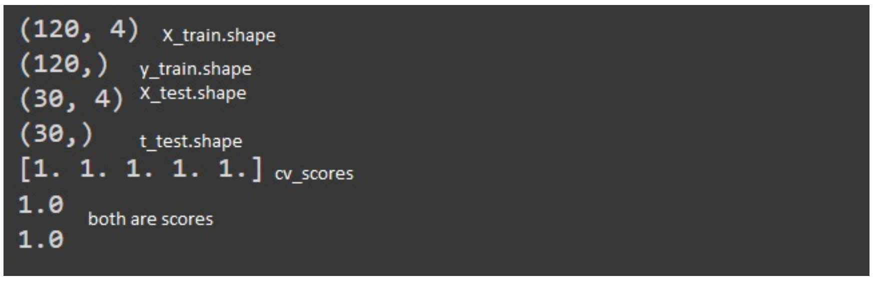
```
0        setosa
1        setosa
2        setosa
3        setosa
4        setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: Iris, Length: 150, dtype: object
```

```
[60] X
```

```
array([[5.1  , 3.5  , 1.4  , 0.2  ],
       [5.2  , 3.   , 1.4  , 0.2  ],
       [5.3  , 3.2  , 1.3  , 0.2  ],
       [5.4  , 3.1  , 1.5  , 0.2  ],
       [5.5  , 3.6  , 1.4  , 0.2  ],
       [5.6  , 3.9  , 1.7  , 0.4  ],
       [5.7  , 3.4  , 1.4  , 0.3  ],
       [5.8  , 3.4  , 1.5  , 0.2  ],
       [5.9  , 2.9  , 1.4  , 0.2  ],
```

As you can see that X, which is the data, has arrays each with 4 columns.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42, stratify=y)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

# Create a k-NN classifier with 4 neighbors: knn
knn = KNeighborsClassifier(n_neighbors = 4)

cv_scores = cross_val_score(knn,X ,y ,cv = 5)

print(cv_scores)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

predictions = knn.predict(X_test)
score_t = knn.score(X_train, y_train)
score = knn.score(X_test, y_test)

# Print the accuracy
print(score_t)
print(score)
```

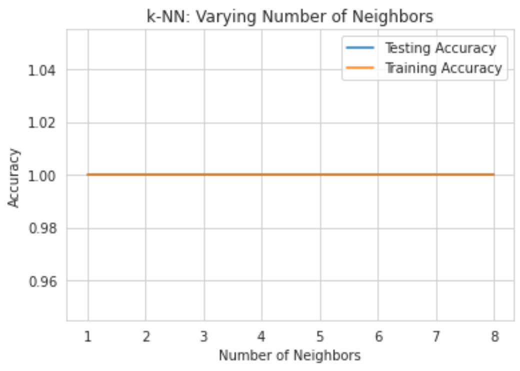


```
(120, 4)
(120,)
(30, 4)
(30,)
[1. 1. 1. 1. 1.]
1.0
1.0
```

With 4D data, 100% accuracy is possible

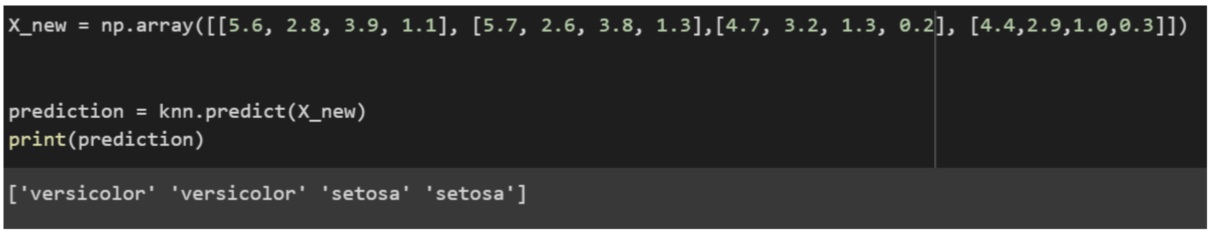
```
# Setup arrays to store train and test accuracies
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over different values of k
for i, k in enumerate(neighbors):
    # Setup a k-NN Classifier with k neighbors: knn
    knn = KNeighborsClassifier(n_neighbors = k)

    # Fit the classifier to the training data
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the testing set
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.title('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

This loop loops over different value of k to find the best k.



At first, I thought I may be dealing with overfitting but as you can see, it is perfect, for both testing and training. But it doesn't seem to be the case.

Here is the model predicting species of the flowers. (More random unseen data)



```
X_new = np.array([[5.6, 2.8, 3.9, 1.1], [5.7, 2.6, 3.8, 1.3],[4.7, 3.2, 1.3, 0.2], [4.4,2.9,1.0,0.3]])


prediction = knn.predict(X_new)
print(prediction)
```

```
['versicolor' 'versicolor' 'setosa' 'setosa']
```

As you can see, the model is working perfectly fine, giving us the perfect answers.

But was it worth it? We just doubled our model complexity. What if we reduce it by half by training and testing with 2D data like we first proposed from the last stage?

Benefits of a simpler model

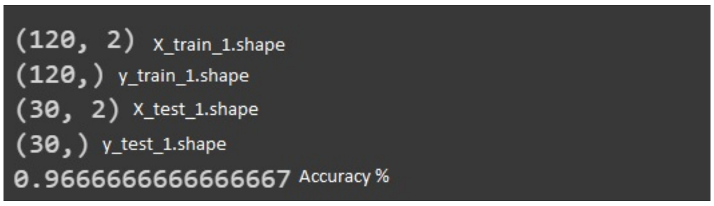
1. **Prevents Overfitting**: A high-dimensional dataset having too many features can sometimes lead to overfitting (model captures both real and random effects).

2. **Interpretability**: An over-complex model having too many features can be hard to interpret especially when features are correlated with each other.
3. **Computational Efficiency**: A model trained on a lower-dimensional dataset is computationally efficient (execution of algorithm requires less computational time).

Training ML model with 2D data

This time I will be using the SVM (Support Vector Machine) algorithm with 2D data, to switch things up a bit. SVM or Support Vector Machine is **a linear model for classification and regression problems**. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

```
y_svm = df['Iris']
X_svm = df.drop(['swcm','slcm','Iris','PetalSize', 'Status'], axis = 1).values
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_svm, y_svm, test_size = 0.2, random_state=42, stratify=y)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

classifier = SVC()
classifier.fit(X_train_1, y_train_1)

y_pred = classifier.predict(X_test_1)
scores_svm = classifier.score(X_test_1, y_test_1)
print(scores_svm)
```



```
(120, 2) X_train_1.shape
(120,) y_train_1.shape
(30, 2) X_test_1.shape
(30,) y_test_1.shape
0.9666666666666667 Accuracy %
```

In X_svm (data), ‘swcm’,’slcm’, and other less features are removed. Everything else stays the same.

You can see in X_train_1’s shape, we’ve gone from 120 rows 4 columns in the previous model to 120 rows 2 columns.

To sum it all up, we have lost only approximately 4% in accuracy, but we reduced our model complexity by half! sounds like bargain trade!

Since we are going to be dealing with just flowers data. And nearly all flowers have only petals and sepals with only exception of some plants that **have one undifferentiated whorl composed of structures called tepals.** The dataset doesn't seem to get more complex than 4 or 5 dimensional. It is highly recommended that we train our potential full scale model without having to cut any feature because even at 5 to 10 features, it is still considered to be acceptable and not highly complex. The computational performance is most likely to be unaffected.

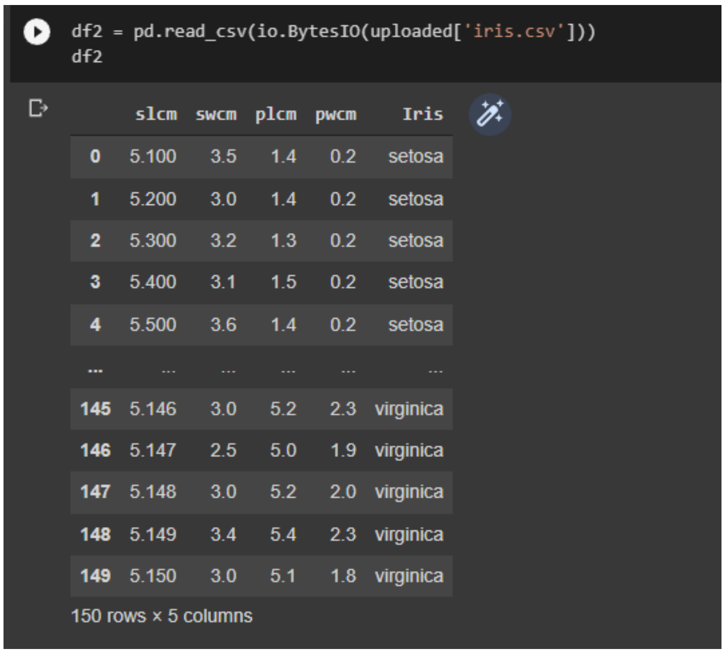
> **Note that K-Nearest Neighbour trained with 2D data also gives out the same accuracy.**

Iris Dataset and unsupervised learning

What if our dataset is missing a target variable? Well, unsupervised learning is the answer.

An unsupervised learning model allows us to visualise the pattern in our dataset. It is **a machine learning technique in which the users do not need to supervise the model!** Which means you do not need a target variable.

I will be using the K-Means Clustering algorithm to showcase Iris Dataset with unsupervised learning. K-Means Clustering is one of many unsupervised learning algorithms. K-means clustering aims to partition data into k clusters in a way that data points in the same cluster are similar and data points in the different clusters are farther apart. Now, let’s see what if our dataset wasn’t given with a target variable. Can we still group them by species?



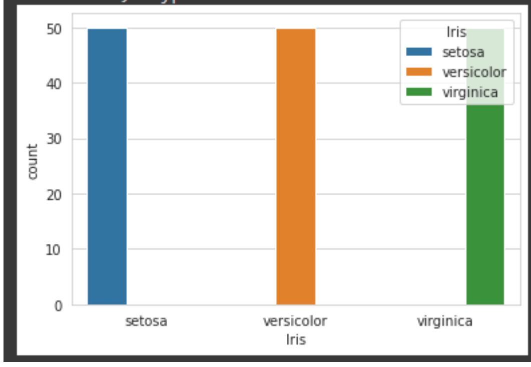
```python
df2 = pd.read_csv(io.BytesIO(uploaded['iris.csv']))
df2
```

	slcm	swcm	plcm	pwcm	Iris
0	5.100	3.5	1.4	0.2	setosa
1	5.200	3.0	1.4	0.2	setosa
2	5.300	3.2	1.3	0.2	setosa
3	5.400	3.1	1.5	0.2	setosa
4	5.500	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	5.146	3.0	5.2	2.3	virginica
146	5.147	2.5	5.0	1.9	virginica
147	5.148	3.0	5.2	2.0	virginica
148	5.149	3.4	5.4	2.3	virginica
149	5.150	3.0	5.1	1.8	virginica

150 rows × 5 columns

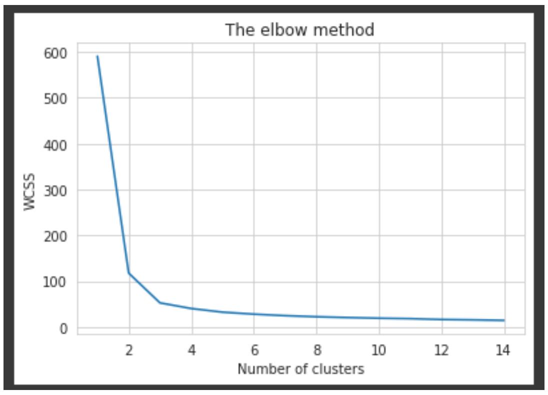
Load up our dataset

```
setosa        50
versicolor    50
virginica     50
Name: Iris, dtype: int64
```



```python
sns.countplot(x ="Iris", hue ="Iris", data = df2)
df2.loc[:,"Iris"].value_counts()
```

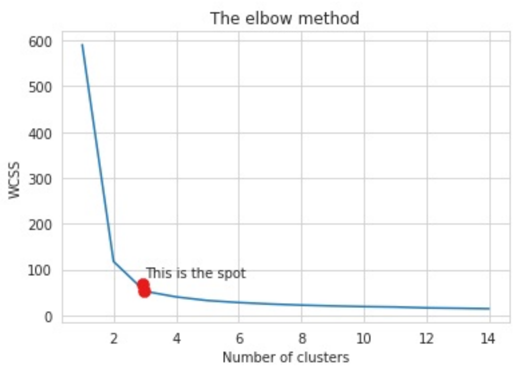
A little bit of visualisation to see if everything’s there, because just this time, where we have the luxury of having a dataset with target variable. In this section, we are going practise for a dataset without target variable, so we have to drop the target variable.



```
#Finding the optimum number of clusters for k-means classification
wcss = []
data_new = df2.drop(["Iris"], axis=1)
for i in range(1, 15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(data_new)
    wcss.append(kmeans.inertia_)
```

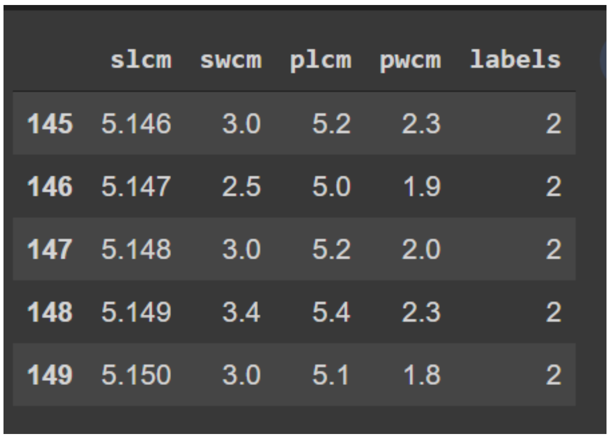
```
plt.plot(range(1, 15), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```

Now, we need to use the elbow method to find the optimum number of clusters for K-Means. In cluster analysis, the elbow method is **a heuristic used in determining the number of clusters in a data set.** In the Elbow method, we are actually varying the number of clusters (K) from 1- 14. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. **When we analyse the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.** So in other words, choose an "elbow" in the WCSS plot, where WCSS begins to decrease more slowly



Moving on,

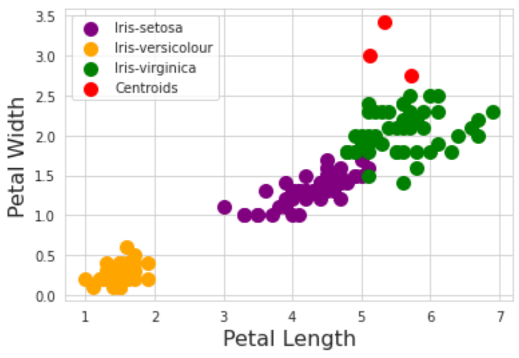
Now, it is time to fit our dataset on to model and get prediction from it. Then, after doing so, I will create new column for the predicted labels. And lastly plot the result.



	slcm	swcm	plcm	pwcm	labels
145	5.146	3.0	5.2	2.3	2
146	5.147	2.5	5.0	1.9	2
147	5.148	3.0	5.2	2.0	2
148	5.149	3.4	5.4	2.3	2
149	5.150	3.0	5.1	1.8	2

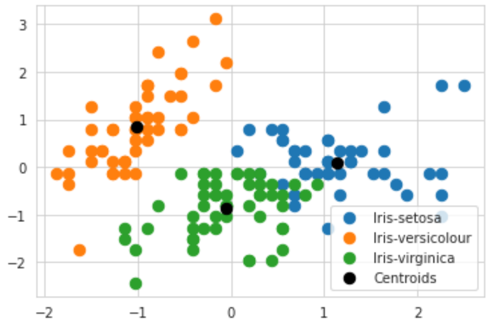
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(data_new)
data_new["labels"] = y_kmeans
data_new.tail()
```

Here's the 2D plot of predicted labels clustered into their own cluster



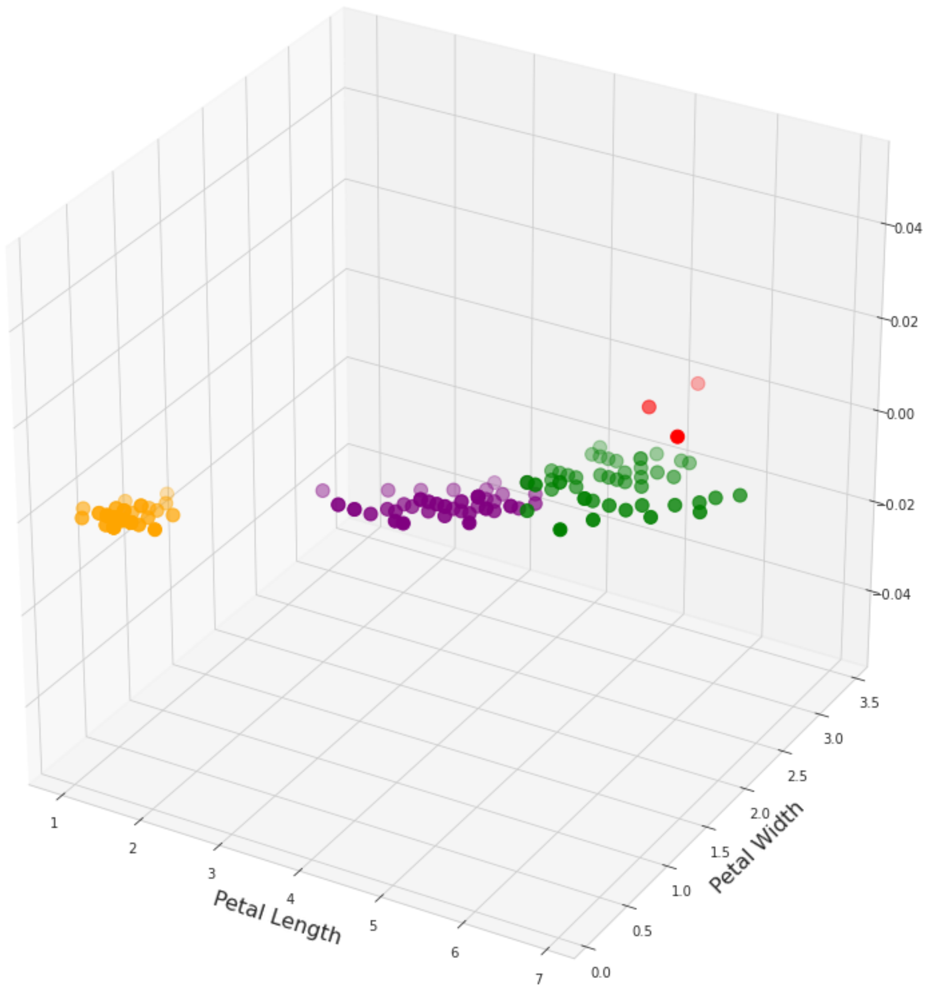
```
#Visualising the clusters
plt.scatter(data_new.plcm[data_new.labels == 0], data_new.pwcm[data_new.labels == 0], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(data_new.plcm[data_new.labels == 1], data_new.pwcm[data_new.labels == 1], s = 100, c = 'orange', label = 'Iris-versicolour
plt.scatter(data_new.plcm[data_new.labels == 2], data_new.pwcm[data_new.labels == 2], s = 100, c = 'green', label = 'Iris-virginica')
plt.xlabel('Petal Length', fontsize=16)
plt.ylabel('Petal Width', fontsize=16)
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.legend()
```

As you can see there are 3 clusters in total which are visualised in different colours, very similar to the scatterplot of our dataset with target variable(species). If you are curious why those centroids aren't any way near their clusters, it is because those centriods are means of **every** measurement but the plot only included petal length and width. So naturally those centroids are going look like they are out of place.



It would have look like this if I didn't only subset for just petal length and width

Here's the 3D plot of the clusters.



```
fig = plt.figure(figsize = (15,15))
ax = fig.add_subplot(111, projection='3d')
plt.scatter(data_new.plcm[data_new.labels == 0], data_new.pwcm[data_new.labels == 0], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(data_new.plcm[data_new.labels == 1], data_new.pwcm[data_new.labels == 1], s = 100, c = 'orange', label = 'Iris-versicolour
plt.scatter(data_new.plcm[data_new.labels == 2], data_new.pwcm[data_new.labels == 2], s = 100, c = 'green', label = 'Iris-virginica')
plt.xlabel('Petal Length', fontsize=16)
plt.ylabel('Petal Width', fontsize=16)

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.show()
```

There isn’t much we can gain from a 3D plot in this scenario, however if you look at the graph again. You will see that these clusters are laying down together and form a sort of a plane. Which is quite neat.

Executive Summary

We started off with **simple collect & store**; Iris dataset from UCI Machine Learning Repository website was downloaded and stored on my harddrive. Things started to warm up in **preparation**; a pattern was discoverd during the “cleaning” of the dataset, Excel was used. Juicy’est part of the report, **EDA & Visualisation;** this was where the study of the dataset begins, the mentioned pattern got revisited and understood. Last but not least, **Machine Learning model;** supervised models were trained with 2D and 4D data, and pattern shown by a trained unsupervised model.

The mentioned pattern is the uniqueness of these Irises petal sizes compared to one another. Setosa has a mean petal length of about 1.5 centimeter and a mean petal width of only 0.2 centimeter. Versicolor, in the middle of the pack, with a mean petal length of 4.2 centimeters and a mean petal width of 1.3 centimeter. Biggest of them all, Virginica, 5.6 centimeters in mean petal length and about 2 centimeters in petal width. This pattern is crucial because it could be used to classify flowers. It has been discovered that, even though there might be a slight overlap in sizes for Versicolor and Virginica, it is minimal. Therefore, we could say that petal length and petal width are two critical features that can be implemented to a machine learning model and would give the result with minimal error.

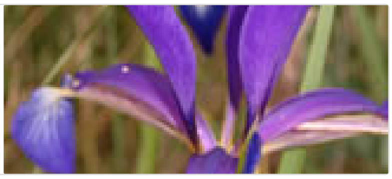
Thus, machine learning models are getting built on top of this pattern. Starting with a highly complex model, K-Nearest Neighbour was used as a learning algorithm, trained by 4D data with 4 features; sepal width and length, and petal width and length. The results given by this model were impressive and intriguing, along with a 100% accuracy. But then, a question was raised, what if a simpler model is possible? but with slightly more errors, although still gives out acceptable results. Thus, a second model was made. Support Vector Machine was used as the learning algorithm, trained by 2D data with 2 features; petal width and length. This only compromised the accuracy by 4%, now, from 100% to about 96%. This result is still highly acceptable because 90% range is still considered to be highly accurate. Moreover, the model complexity was reduced by half. In light of these facts, it is now certain that these models are working perfectly fine, and the experiment was a success. Furthermore, no hyperparameter tuning was done as there was no reason to do so. One last thing, unsupervised learning with K-Means clustering with the same dataset but without target variable. This one was an experiment inspired by curiosity; wanting to know what unsupervised learning looks like in action. And to practise for dataset without target variable. Procedure is familiar like supervised learning but without ‘y’ or target output. K-Means clustering allows us to visualise the pattern of the data. During the training the algorithm, calculated using the elbow method, 3 clusters is the perfect number of k (3 species, so it makes sense). When the model is trained, the predicted result (only subset to just petal length and width) was plot, it was nearly identical to its rightfully counterpart, an earlier scatterplot plotted with the same dataset but with target variable (also subset to just petal length and width). Thus, it is certain that the algorithm did its job perfectly.

This project has been a journey filled with lessons. There were two goals set in the beginning of this project. First, the technical goal, to make a working machine learning model that predicts species of flowers, the process was already explained above. The second goal, to treat this project as an excursion, an educational trip outside the classroom. A journey to learn and ascertain one’s knowledge on moving parts behind these learning algorithms, sharpen one’s data analysis skill; whether cleaning, visualising, or exploring skills. This project can help students who are eager to learn and want to feel more confident in their data analysis skills. They can follow the guides in the ‘main event’ section and do this project themselves, with their own twists. Or simply just read it. The project offers a little bit of everything for everyone, it is the intention of the author.

Bibliography

Multivariate Number of Instances: 150 Area: Life Attribute Characteristics: Real Number of Attributes: 4 Date Donated 1988-07-01 Associated Tasks: Classification Missing Values? No Number of Web Hits: 4522153 Source: Creator: R.A. Fisher Donor: Michael Marshall (MARSHALL% PLU '@' io.arc.nasa.gov) Data Set Information: This is perhaps the

https://archive.ics.uci.edu/ml/datasets/iris



Iris virginica - Wikipedia

Iris virginica , with the common name Virginia iris, is a perennial species of flowering plant, native to eastern North America. It is common along the coastal plain from Florida to Georgia in the Southeastern United States.

W https://en.wikipedia.org/wiki/Iris_virginica

Iris versicolor - Wikipedia

It is a species of native to North America, in the Eastern United States and Eastern Canada. It is common in sedge meadows, marshes, and along streambanks and shores. The specific epithet versicolor means "variously coloured".

W https://en.wikipedia.org/wiki/Iris_versicolor

Iris setosa - Wikipedia

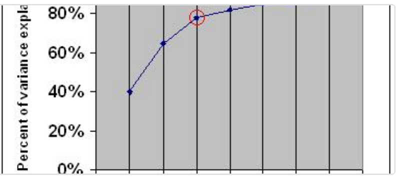
Iris setosa , the bristle-pointed iris, is a species of flowering plant in the genus of the family Iridaceae, it belongs the subgenus and the series Tripetalae . It is a rhizomatous perennial from a wide range across the Arctic sea, including Alaska, Maine, Canada (including British Columbia, Newfoundland, Quebec and Yukon), Russia (including Siberia),

W https://en.wikipedia.org/wiki/Iris_setosa

Elbow method (clustering) - Wikipedia

In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

https://en.wikipedia.org/wiki/Elbow_method_(clustering)



SVM | Support Vector Machine Algorithm in Machine Learning

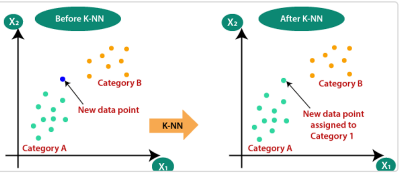
Explanation of support vector machine (SVM), a popular machine learning algorithm or classification Implementation of SVM in R and Python Learn about the pros and cons of Support Vector Machines(SVM) and its different applications Mastering machine learning algorithms isn't a myth at all. Most beginners start by learning regression.

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning



K-Means Clustering Explained - neptune.ai

Clustering was introduced in 1932 by H.E. Driver and A.L.Kroeber in their paper on "Quantitative expression of cultural relationship". Since then this technique has taken a big leap and has been used to discover the unknown in a number of application areas eg. Healthcare. Clustering is a type of unsupervised learning where the references need

https://neptune.ai/blog/k-means-clustering