Why we choose KNN, SVM:

KNN: it used for both Classification models that used for categorical and regression used for numerical. Because iris is regression continuous dataset KNN was the best choice for it.
SVM: because there was a clear margin of separation between classes, very helpful when there are more dimensions than samples, can work with numerical continuous dataset very well and relatively effective with memory

Steps for building ML models:

1. Choose the problems and ask the right questions.

We identify what kind of problems we want to work on. So, we can choose data. Through collection or choosing ready-to-download data, we ask questions related to the problem we want to solve using this data.

2. choose the data:

we choose iris data. Because Sklearn library includes the Iris Dataset. Iris is one of the many datasets available in Sklearn to explore machine learning techniques.

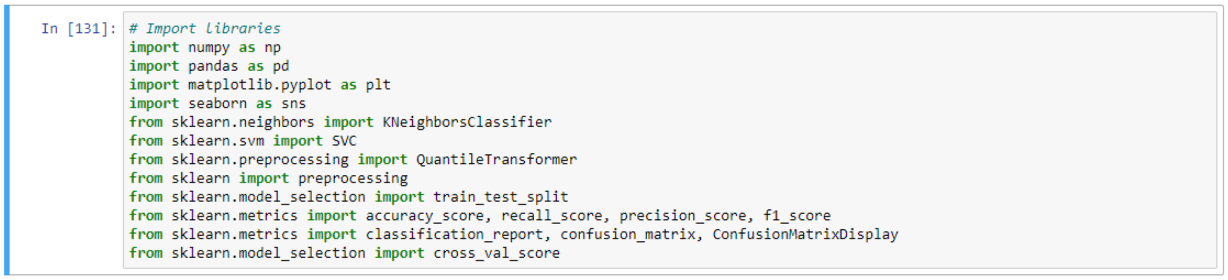
Iris has a tri-class target variable and four numerical attributes. Both clustering and classification may be done with this dataset. Iris is the "hello world" of machine learning, according to data scientists. Let's learn how to load and explore the well-known dataset of iris plant species.

Sepal length, sepal width, petal length, and petal width are the four features in this dataset, and the target variable has three classes: "setosa," "versicolor," and "virginica." Predicting the target class from the values of the four features is the goal of a multiclass classifier.

3. Data preparation:

Once choosing, the data needs to be prepared for the next step. Data preparation is the phase of placing data in an appropriate location and preparing it for use in machine learning training. to understand the nature of the data that we must work with. we need to understand the nature, format, and quality of the data. A more accurate grasp of the data results in successful results.

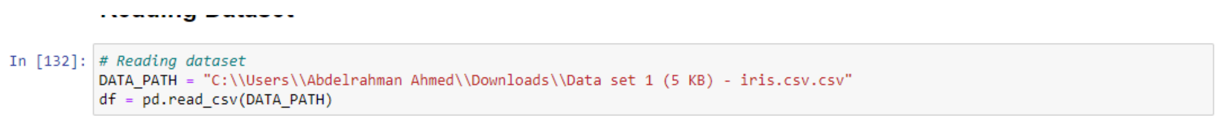
3.1. import the library:



```
In [131]: # Import Libraries
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.svm import SVC
          from sklearn.preprocessing import QuantileTransformer
          from sklearn import preprocessing
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
          from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
          from sklearn.model_selection import cross_val_score
```

In this step we choose the library that will help us, numpy library is a Python package that is used to manipulate arrays. Additionally, it has matrices, Fourier transform, and functions for working in the area of linear algebra. Panda library is capable of quickly loading data from various databases and data formats: can be utilized with a wide variety of data types. Matplotlib provides a variety of graphical displays, including Bar Graphs, Histograms, Line Graphs, Scatter Plots, Stem Plots, etc. seaborn Clearly visualise our data on a plot. We can use this library to visualise our data without having to worry about the inside workings; all we have to do is feed our data set or data inside the relplot() function, and it will compute and place the value appropriately.sklearn It's a versatile, user-friendly tool that can create neuroimages and anticipate consumer behaviour, among other things. It is cost-free and simple to use.

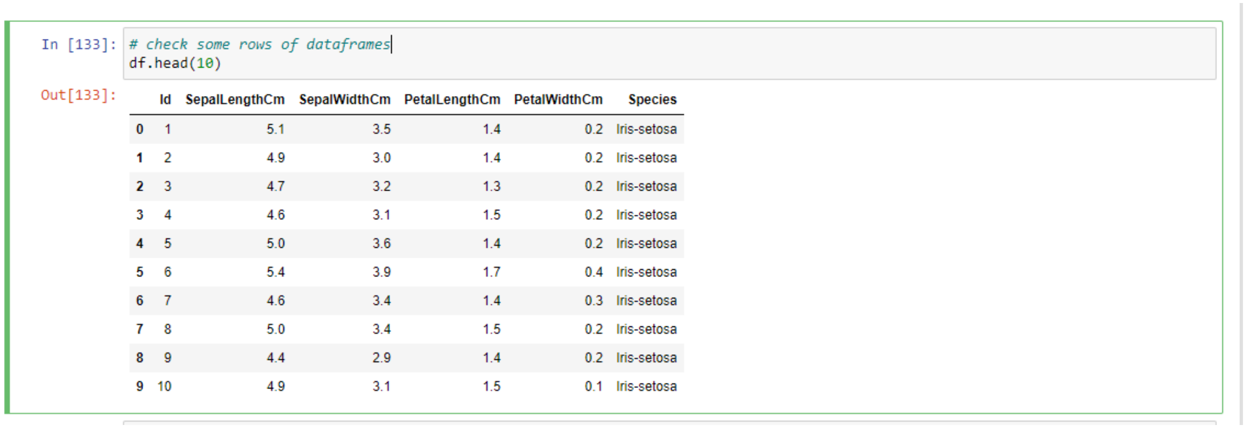
3.2. Reading the data.



```
In [132]: # Reading dataset
          DATA_PATH = "C:\\Users\\Abdelrahman Ahmed\\Downloads\\Data set 1 (5 KB) - iris.csv.csv"
          df = pd.read_csv(DATA_PATH)
```

In this step we load csv file using the above code, access data from csv file and retrieves data in the form of data frame.

3.3. check some rows of dataframes:


```
In [133]: # check some rows of dataframes
          df.head(10)
```

Out[133]:

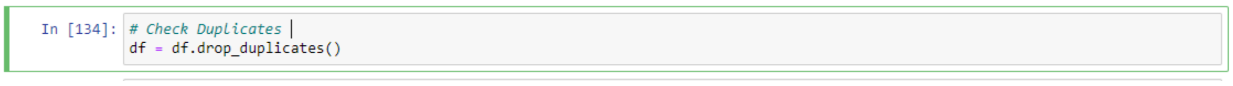
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

This code retrieve the first number of the rows.

4. Data Wrangling:

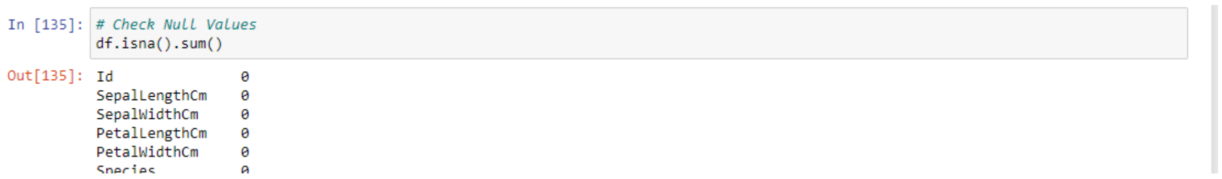
It's the most important step in the analysis because if there are any mistakes in this data, the analysis will be wrong. The process of cleaning the data and converting it into a usable format, choosing which variables to use, transforming the data into the correct format, and making it more suitable for analysis in the next step

4.1. Check Duplicates:


```
In [134]: # Check Duplicates
          df = df.drop_duplicates()
```

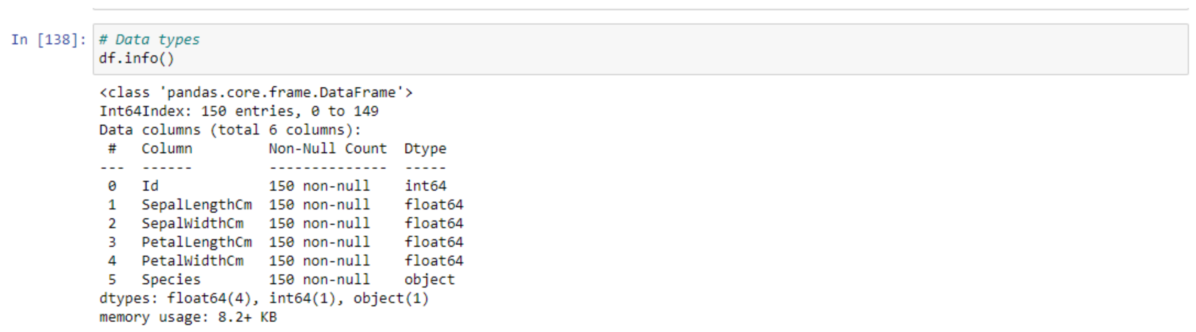
In this code, we check if there is duplicate data and order python to delete it.

4.2. Check Null values


```
In [135]: # Check Null Values
          df.isna().sum()

Out[135]: Id               0
          SepalLengthCm    0
          SepalWidthCm     0
          PetalLengthCm    0
          PetalWidthCm     0
          Species          0
```

This code to see if there is empty values .

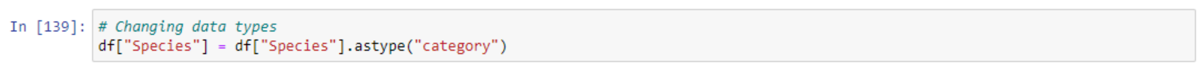
4.3. Data types



```
In [138]: # Data types
          df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 8.2+ KB
```

In the above code we retrieve the info about all the data in the csv file.

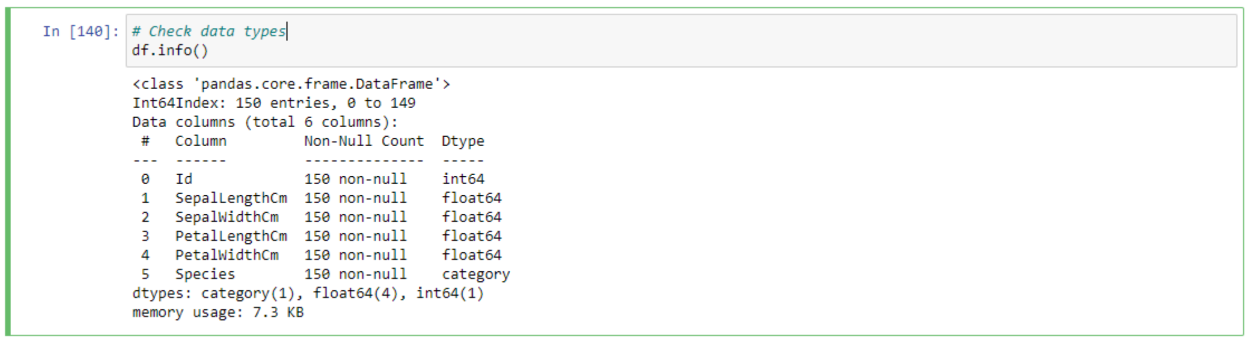
4.4. Changing data types



```
In [139]: # Changing data types
          df["Species"] = df["Species"].astype("category")
```

In the above code we change species from object to categories.

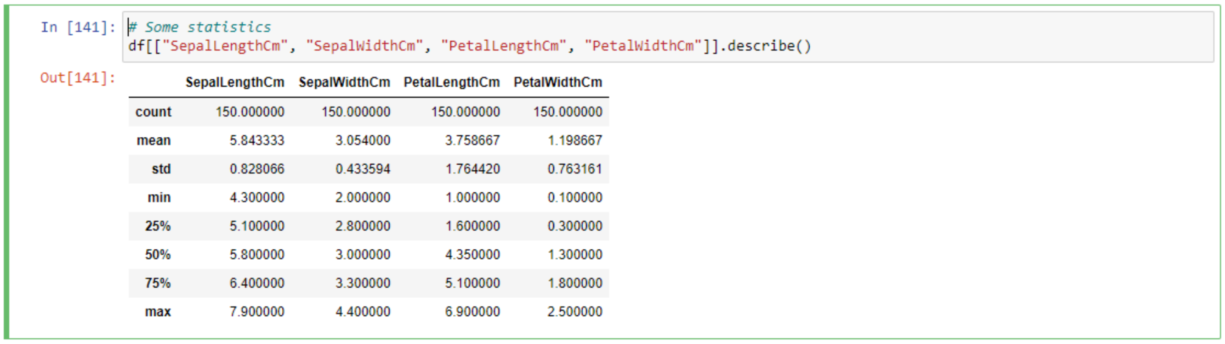
4.5. Check data types



```
In [140]: # Check data types
          df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   Id             150 non-null    int64   
 1   SepalLengthCm  150 non-null    float64 
 2   SepalWidthCm   150 non-null    float64 
 3   PetalLengthCm  150 non-null    float64 
 4   PetalWidthCm   150 non-null    float64 
 5   Species        150 non-null    category
dtypes: category(1), float64(4), int64(1)
memory usage: 7.3 KB
```

In this code we want to make sure that the species types change.

4.6. Some statistics



```
In [141]: # Some statistics
          df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]].describe()
```

Out[141]:

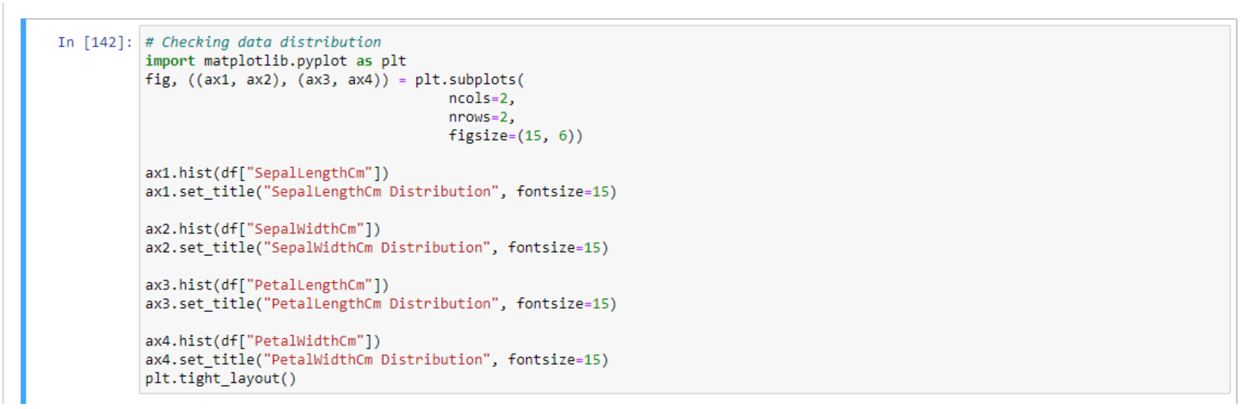
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In the above code we made some statics for 4 columns.

5. Data Analysis:

After cleaning the data, we start to analyse the data through select machine learning techniques such as classification, regression, cluster analysis, association, etc. Then use the prepared data to build and evaluate the model.

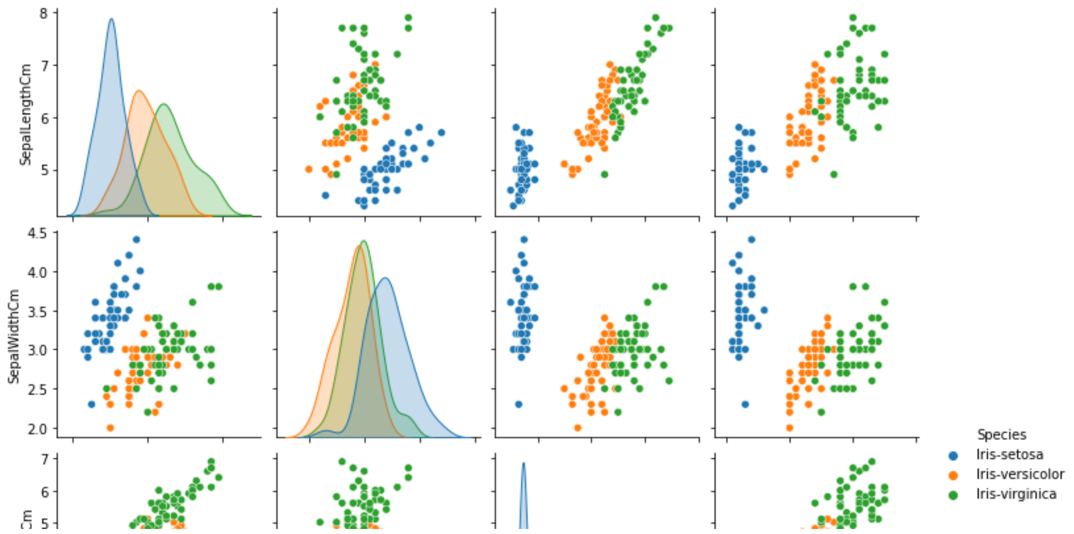
5.1. Checking data distribution



```
In [142]: # Checking data distribution
          import matplotlib.pyplot as plt
          fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(
                                          ncols=2,
                                          nrows=2,
                                          figsize=(15, 6))

          ax1.hist(df["SepalLengthCm"])
          ax1.set_title("SepalLengthCm Distribution", fontsize=15)

          ax2.hist(df["SepalWidthCm"])
          ax2.set_title("SepalWidthCm Distribution", fontsize=15)

          ax3.hist(df["PetalLengthCm"])
          ax3.set_title("PetalLengthCm Distribution", fontsize=15)

          ax4.hist(df["PetalWidthCm"])
          ax4.set_title("PetalWidthCm Distribution", fontsize=15)
          plt.tight_layout()
```

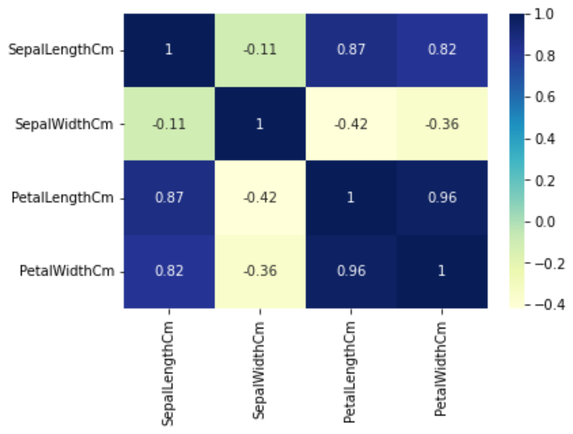
5.2. Checking the correlation pairplot:

```
In [143]: # Checking the correlation pairplot
          sns.pairplot(df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",
                           "PetalWidthCm", "Species"]], hue="Species")

Out[143]: <seaborn.axisgrid.PairGrid at 0x2d82d6ca7c0>
```



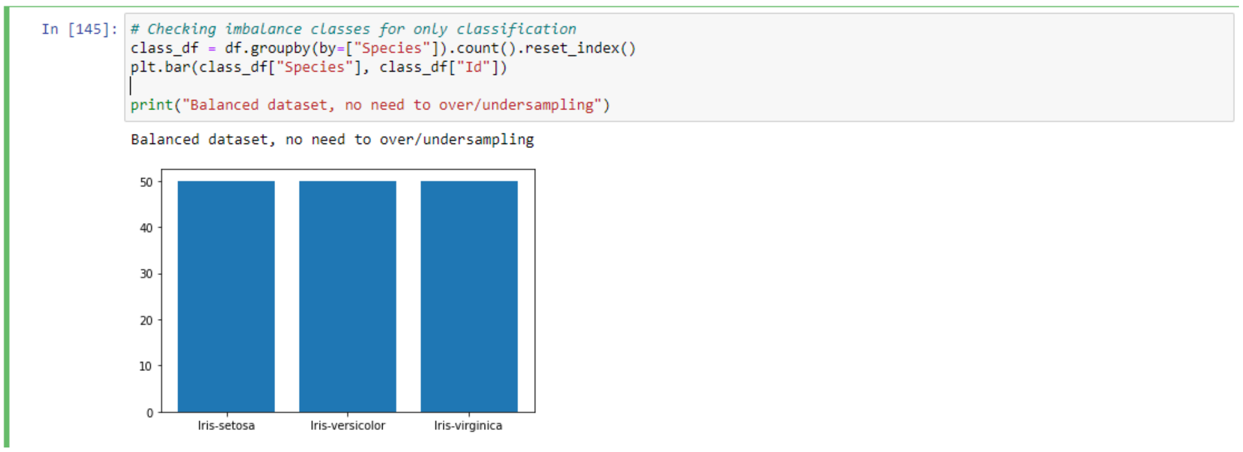
5.3. Correlation heatmap:

```
In [144]: # Correlation heatmap
          corr_df = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",
                        "PetalWidthCm"]].corr()
          dataplot = sns.heatmap(corr_df, cmap="YlGnBu", annot=True)
```



	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1	-0.11	0.87	0.82
SepalWidthCm	-0.11	1	-0.42	-0.36
PetalLengthCm	0.87	-0.42	1	0.96
PetalWidthCm	0.82	-0.36	0.96	1

5.4. Checking imbalance classes for only classification

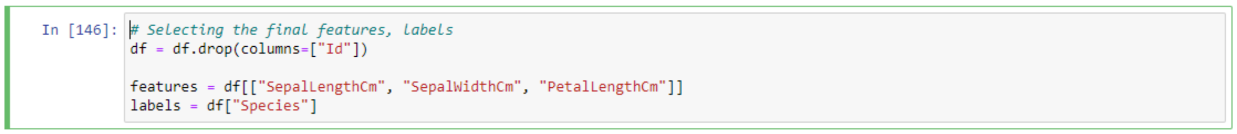


```
In [145]: # Checking imbalance classes for only classification
          class_df = df.groupby(by=["Species"]).count().reset_index()
          plt.bar(class_df["Species"], class_df["Id"])

          print("Balanced dataset, no need to over/undersampling")

          Balanced dataset, no need to over/undersampling
```

6. Train Model:

Through the dataset, we choose different algorithms to train the data to have the result we want.

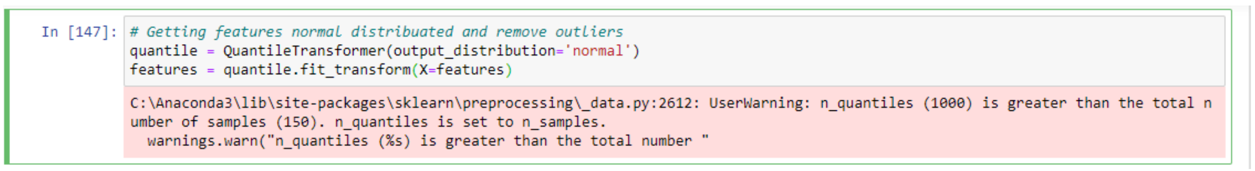
6.1. Selecting the final features, labels



```
In [146]: # Selecting the final features, labels
          df = df.drop(columns=["Id"])

          features = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm"]]
          labels = df["Species"]
```

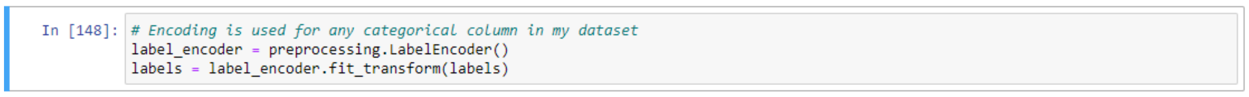
In the above code we saw that id columns not importance. so, we give order to delete this column. And we give order to use the rest of columns.

6.2. Getting features normal distributed and remove outliers



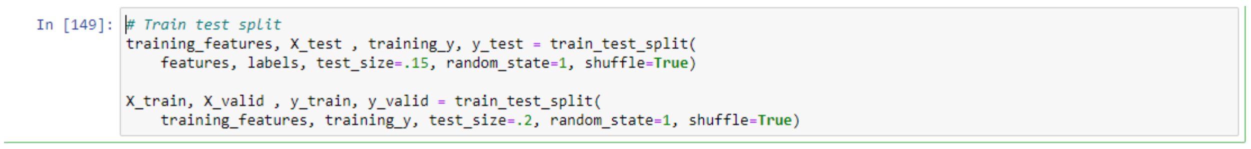
```
In [147]: # Getting features normal distributed and remove outliers
          quantile = QuantileTransformer(output_distribution='normal')
          features = quantile.fit_transform(X=features)

          C:\Anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:2612: UserWarning: n_quantiles (1000) is greater than the total n
          umber of samples (150). n_quantiles is set to n_samples.
            warnings.warn("n_quantiles (%s) is greater than the total number "
```

6.3. Encoding is used for any categorical column in my dataset

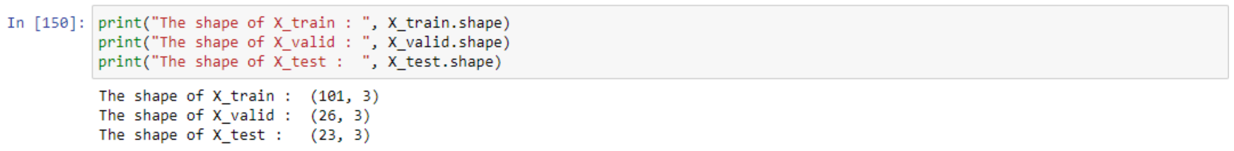


```
In [148]: # Encoding is used for any categorical column in my dataset
          label_encoder = preprocessing.LabelEncoder()
          labels = label_encoder.fit_transform(labels)
```

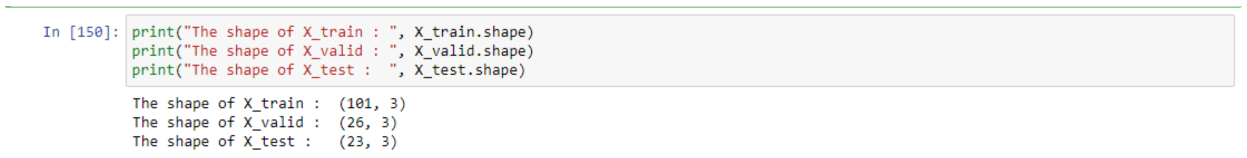
6.4. Train test split



```
In [149]: # Train test split
          training_features, X_test , training_y, y_test = train_test_split(
              features, labels, test_size=.15, random_state=1, shuffle=True)

          X_train, X_valid , y_train, y_valid = train_test_split(
              training_features, training_y, test_size=.2, random_state=1, shuffle=True)
```



```
In [150]: print("The shape of X_train : ", X_train.shape)
          print("The shape of X_valid : ", X_valid.shape)
          print("The shape of X_test :  ", X_test.shape)

          The shape of X_train :  (101, 3)
          The shape of X_valid :  (26, 3)
          The shape of X_test :   (23, 3)
```

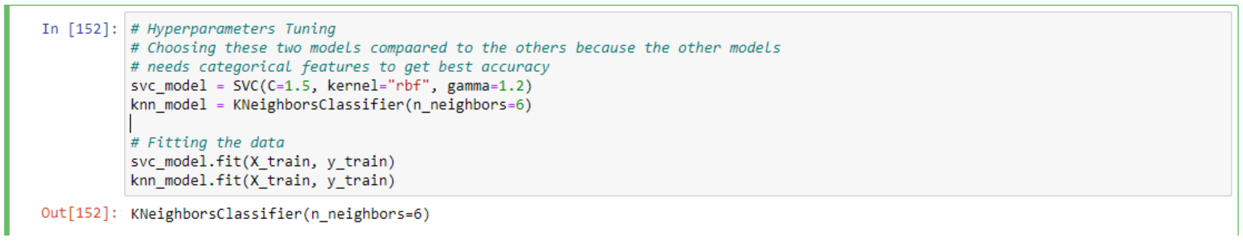


```
In [150]: print("The shape of X_train : ", X_train.shape)
          print("The shape of X_valid : ", X_valid.shape)
          print("The shape of X_test :  ", X_test.shape)

          The shape of X_train :  (101, 3)
          The shape of X_valid :  (26, 3)
          The shape of X_test :   (23, 3)
```

7. Test Model:

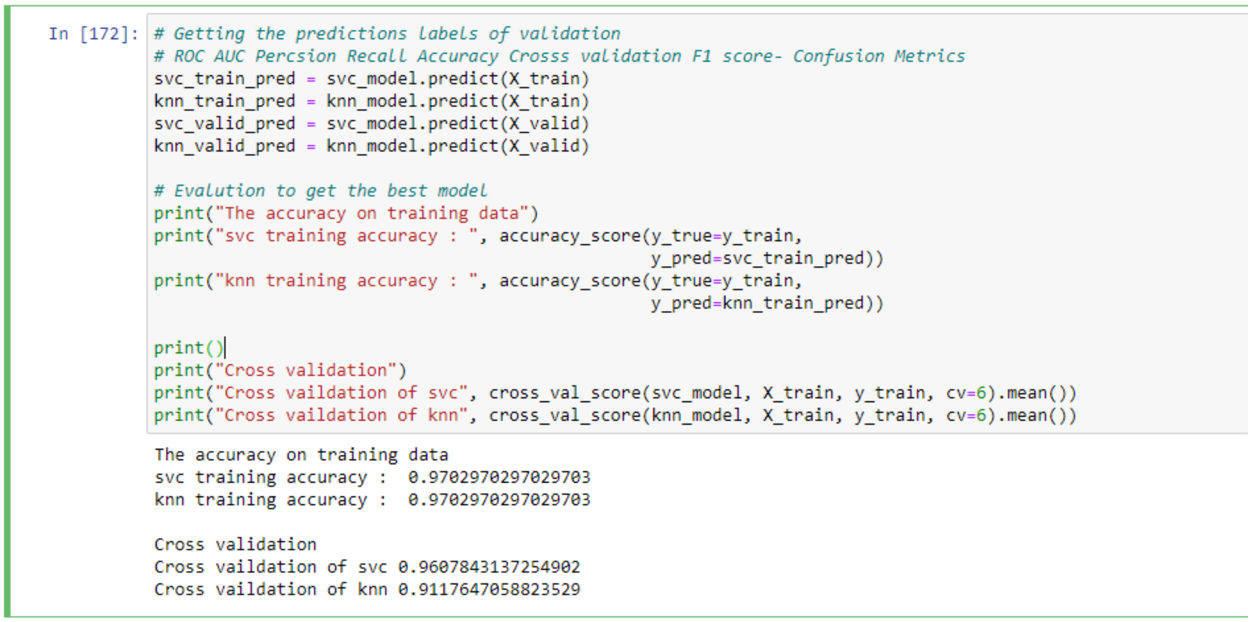
After we finish training our model, we test the model. Check the model's accuracy by providing it with a test dataset.

7.1. Fitting the data



```
In [152]: # Hyperparameters Tuning
          # Choosing these two models compaared to the others because the other models
          # needs categorical features to get best accuracy
          svc_model = SVC(C=1.5, kernel="rbf", gamma=1.2)
          knn_model = KNeighborsClassifier(n_neighbors=6)

          # Fitting the data
          svc_model.fit(X_train, y_train)
          knn_model.fit(X_train, y_train)

Out[152]: KNeighborsClassifier(n_neighbors=6)
```

7.2. Evalution to get the best model

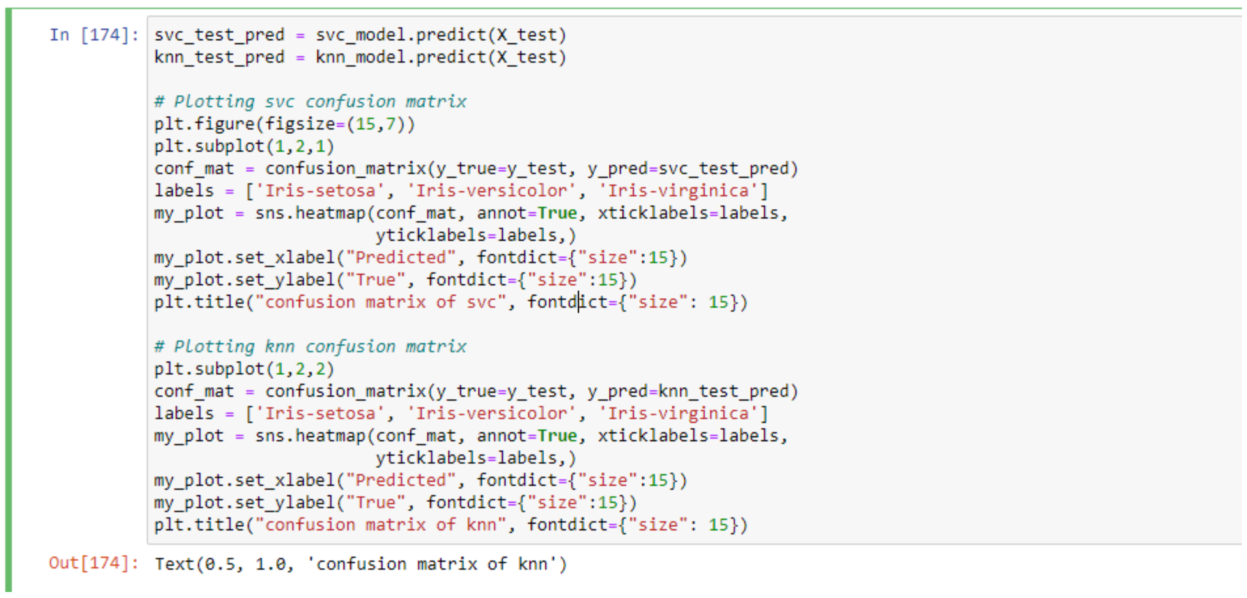


```
In [172]: # Getting the predictions labels of validation
          # ROC AUC Percsion Recall Accuracy Crosss validation F1 score- Confusion Metrics
          svc_train_pred = svc_model.predict(X_train)
          knn_train_pred = knn_model.predict(X_train)
          svc_valid_pred = svc_model.predict(X_valid)
          knn_valid_pred = knn_model.predict(X_valid)

          # Evalution to get the best model
          print("The accuracy on training data")
          print("svc training accuracy : ", accuracy_score(y_true=y_train,
                                                           y_pred=svc_train_pred))
          print("knn training accuracy : ", accuracy_score(y_true=y_train,
                                                           y_pred=knn_train_pred))

          print()
          print("Cross validation")
          print("Cross vaildation of svc", cross_val_score(svc_model, X_train, y_train, cv=6).mean())
          print("Cross vaildation of knn", cross_val_score(knn_model, X_train, y_train, cv=6).mean())

          The accuracy on training data
          svc training accuracy :  0.9702970297029703
          knn training accuracy :  0.9702970297029703

          Cross validation
          Cross vaildation of svc 0.9607843137254902
          Cross vaildation of knn 0.9117647058823529
```

8. Deployment:

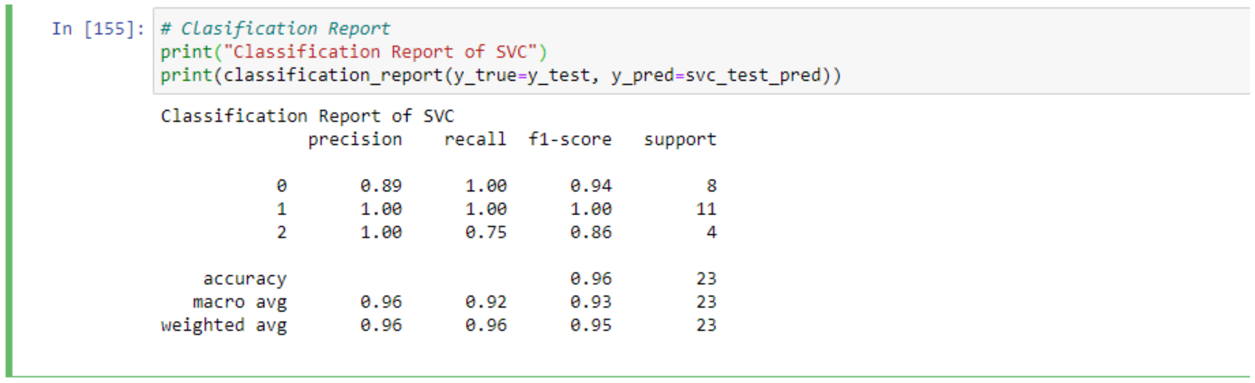
It’s the last step in ML. If the model we used produces an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. However, before implementing a project, we will use the available data to see if it improves its performance or not. The deployment phase is like making the final report for a project.

8.1. Plotting svc.knn confusion matrix

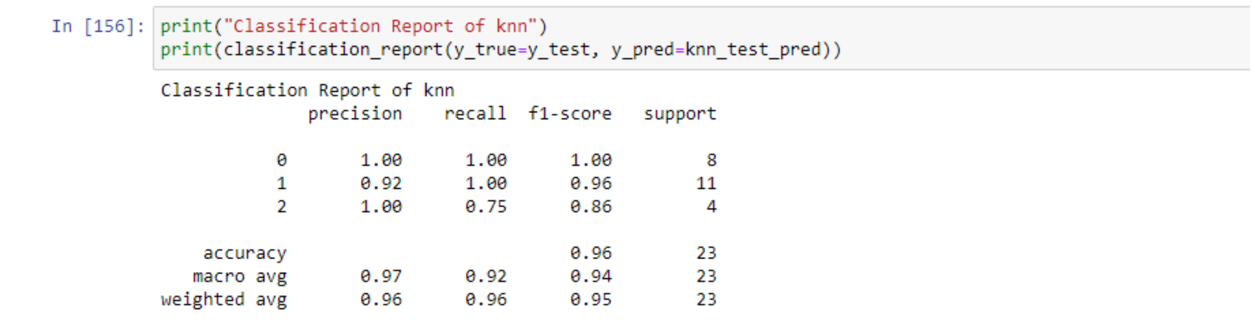


```
In [174]: svc_test_pred = svc_model.predict(X_test)
          knn_test_pred = knn_model.predict(X_test)

          # Plotting svc confusion matrix
          plt.figure(figsize=(15,7))
          plt.subplot(1,2,1)
          conf_mat = confusion_matrix(y_true=y_test, y_pred=svc_test_pred)
          labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
          my_plot = sns.heatmap(conf_mat, annot=True, xticklabels=labels,
                                yticklabels=labels,)
          my_plot.set_xlabel("Predicted", fontdict={"size":15})
          my_plot.set_ylabel("True", fontdict={"size":15})
          plt.title("confusion matrix of svc", fontdict={"size": 15})

          # Plotting knn confusion matrix
          plt.subplot(1,2,2)
          conf_mat = confusion_matrix(y_true=y_test, y_pred=knn_test_pred)
          labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
          my_plot = sns.heatmap(conf_mat, annot=True, xticklabels=labels,
                                yticklabels=labels,)
          my_plot.set_xlabel("Predicted", fontdict={"size":15})
          my_plot.set_ylabel("True", fontdict={"size":15})
          plt.title("confusion matrix of knn", fontdict={"size": 15})

Out[174]: Text(0.5, 1.0, 'confusion matrix of knn')
```

8.2. Classification Report



```
In [155]: # Clasification Report
          print("Classification Report of SVC")
          print(classification_report(y_true=y_test, y_pred=svc_test_pred))

          Classification Report of SVC
                        precision    recall  f1-score   support

                     0       0.89      1.00      0.94         8
                     1       1.00      1.00      1.00        11
                     2       1.00      0.75      0.86         4

              accuracy                           0.96        23
             macro avg       0.96      0.92      0.93        23
          weighted avg       0.96      0.96      0.95        23
```



```
In [156]: print("Classification Report of knn")
          print(classification_report(y_true=y_test, y_pred=knn_test_pred))

          Classification Report of knn
                        precision    recall  f1-score   support

                     0       1.00      1.00      1.00         8
                     1       0.92      1.00      0.96        11
                     2       1.00      0.75      0.86         4

              accuracy                           0.96        23
             macro avg       0.97      0.92      0.94        23
          weighted avg       0.96      0.96      0.95        23
```

It appears from the results that SVC is a bit better than KNN