**DOCUMENT THE MACHINE LEARNING MODEL DEPLOYMENT PROJECT**

TO DEVELOP A MACHINE LEARNING MODEL WITH IBM CLOUD WASTON STUDIO

Machine learning has become an integral part of various industries, from healthcare and finance to marketing and manufacturing. With the advancements in technology, deploying machine learning models has become easier and more efficient. One platform that offers a seamless experience in deploying machine learning models is IBM Watson Studio.

You can follow the below steps:

* **CREATE AN IBM ACCOUNT**: If you don't already have one, sign up for an IBM Cloud account.

* **SET UP WATSON STUDIO**: Log in to your IBM Cloud account and create a Watson Studio service instance. You can find Watson Studio in the IBM Cloud catalog.

* **CREATE OR IMPORT A PROJECT:** Inside Watson Studio, create a new project or import an existing one. A project is where you'll organize your machine learning assets.

* **ADD DATA**: Upload and prepare your data within your project. You can use the Data Refinery tool in Watson Studio to clean and shape your data.

* **TRAIN YOIR MODEL**: Use Watson Studio's tools and Jupyter notebooks to develop and train your machine learning model. You can utilize various libraries and frameworks like scikit-learn or TensorFlow.

* **SAVE AND VERSION YOUR MODEL**: Once your model is trained and tested, save it within your project. Version control is crucial to keep track of model changes.

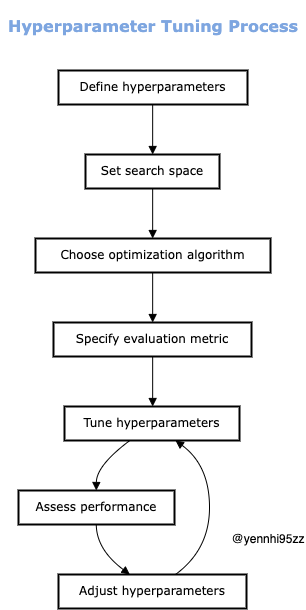
* **CREATE A DEPLOYMENT SPACE**: In Watson Studio, you can create a deployment space. This is where you'll deploy your model for production use.

* **DEPLOY YOUR MODEL**: Within the deployment space, you can deploy your model as an API or batch job. Follow the wizard to configure deployment settings, including hardware resources and scaling options.

* **TEST TOUR DEPLOYMENT:** After deployment, test your model to ensure it's working as expected. You can do this from Watson Studio or by using API calls.
* **MONITOR AND MANAGE**:Watson Studio provides monitoring and management tools to track the performance of your deployed model. You can also update and re-deploy as needed.

* **ACCESS YOUR MODEL**: Once deployed, your model will have an endpoint that you can use to integrate it into your applications or services.The access page actions are the actions that users can take when they visit the access page for the deployed model. The default access page actions are:
* **MAKE PREDICTIONS**: Users can make predictions using the deployed model.
* **VIEW MODEL DETAILS**: Users can view information about the deployed model, such as the model type, the training data, and the model performance.
* **DOWNLOAD MODEL**: Users can download the deployed model. You can add, remove, or modify the access page actions to meet your specific needs.

**CONSIDER EXPERIMENTING WITH ENSEMBLE METHODS OR HYPERPARAMETER TUNING TO OPTIMIZE THE MODEL’S PERFORMANCE**

**STEPS TO PERFORM HYPERPARAMETER TUNING**

1. **SELECT HYPERPARAMETERS TO TUNE:** Different algorithms have different hyperparameters. Determining the correct ones for the chosen algorithm is the first step.
2. **CHOOSE A SEARCH SPACE**: This is the range of values each hyperparameter can take. The larger the search space, the more options the match will consider.
3. **OPTIMIZATION TECHNIQUES**: There are several techniques available, each with its own approach. Including:

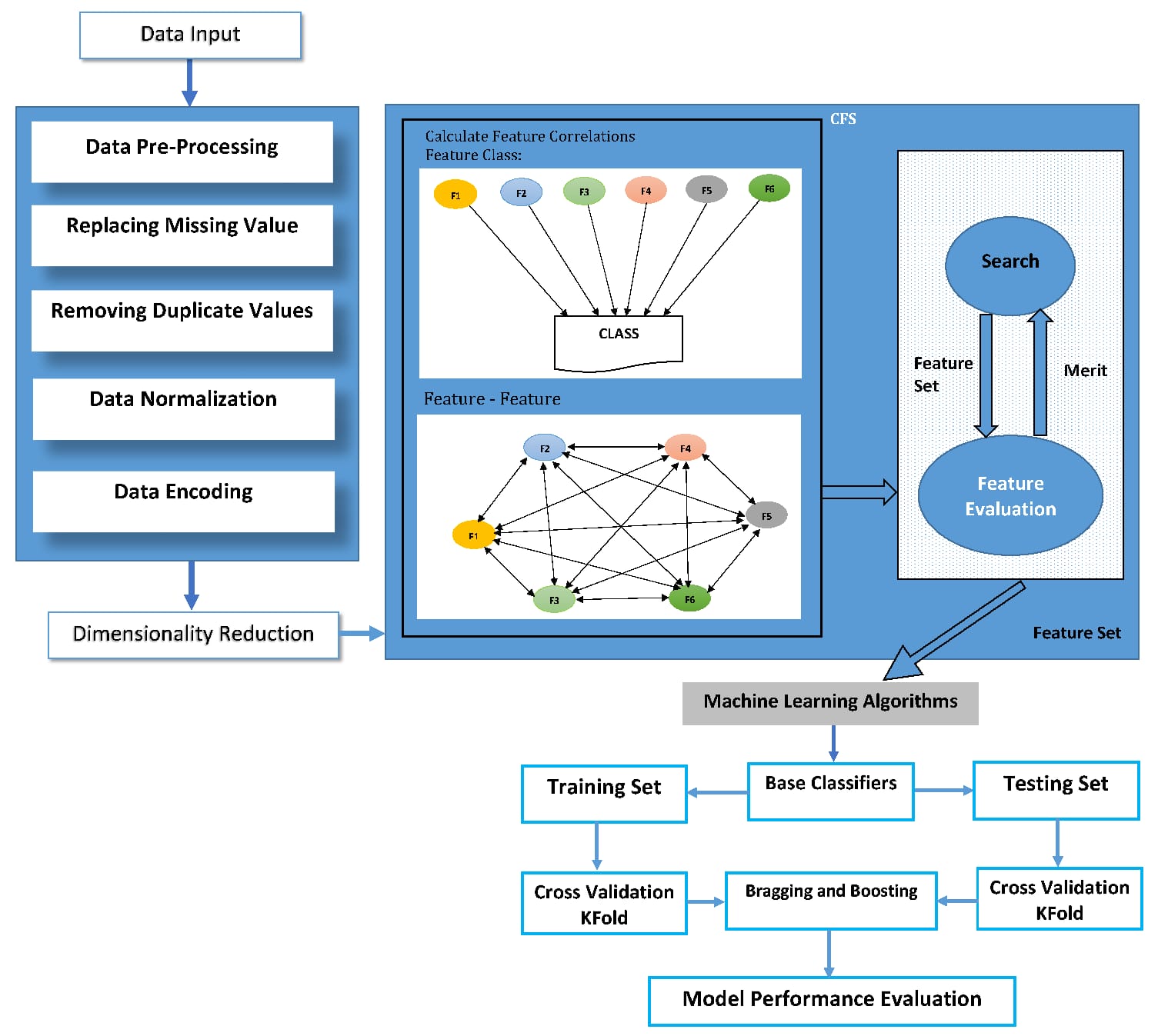
* **Manual Search**: Manually try different hyperparameter values. Simple, but time consuming.
* **Random Search**: Random samples from the search space. Efficient, but may miss optimal values.
* **Grid Search**: Systematically explore all possible combinations. Complete, but computationally expensive.
* **Bayesian Optimization**: Use previous reviews to make informed decisions about where to look next. Efficient and effective.
* **Genetic Algorithms** : Inspired by natural selection, better sets of hyperparameters evolve over generations.

4**. EVALUATE PERFORMANCE**: For each set of hyperparameters, measure the model’s performance on the validation dataset using metrics such as accuracy, precision, or recall.

5**. SELECT BEST HYPERPARAMETERS**: Choose the set of hyperparameters that lead to the best model performance.

**INFLUENCE OF HYPERPARAMETERS ON MODELS:**

Before a performance, visualize a symphony orchestra tuning its instruments. Hyperparameters function similarly to how each instrument's tuning impacts the overall harmony when fine-tuning a machine learning model. Inaccurate hyperparameters can make a model difficult to play, just like an out-of-tune violin can ruin the tone.



Let’s take a closer look at some essential hyperparameters and their influence on shaping the behavior of the model.

1. **TRAIN-TEST SPLIT ESTIMATOR:**

It's vital to talk about the first stage, the training-test split estimator, before delving into the area of machine learning-specific hyperparameters. Although this is not a hyperparameter in the conventional sense, it has an impact on how the model learns. We need data to train a model and data to verify its effectiveness when we are training a model. We divide our dataset into these two halves with the aid of the training test split estimator.

For example using the train\_test\_split function, for instance, we could divide our data into 60% for training and 40% for testing. Consistency in model evaluation is supported by the random\_state parameter's guarantee that the same set of data is consistently generated. Without this control, evaluating a model can turn into a challenging puzzle, and neglecting the random state might cause the model to behave in an unanticipated way. In essence, random\_state acts as the random number generator's seed, stabilizing the model's behavior.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=0)

1. **LOGISTIC REGRESSION CLASSIFIER:**

When we’re talking about classifying things, one common go-to is the Logistic Regression Classifier. Inside its workings, there’s a special knob called C, and it’s connected to something called the ‘regularization parameter,’ let’s call it λ (that’s a Greek letter “lambda”).

Now picture it as adjusting the brake and gas pedals on an automobile. Increasing C is equivalent to depressing the gas pedal more firmly while easing up on the brake. This 'C' aids in regulating how closely the model should resemble the data. If C is turned up too high, the model may overfit the data by memorizing it too thoroughly, but if C is kept low, the model may underfit the data by failing to recognize its patterns. Finding the ideal Ci is comparable to finding the ideal balance between safe driving and driving quickly.

Mathematically: C = 1/λ

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(C=1000.0, random\_state=0)

1. **K-NEAREST NEIGHBOR(KNN)CLASSIFIER:**

Choosing the ideal number of neighbors and the power parameter p are key components of the KNN algorithm. How many data points are taken into account while making predictions is controlled by the n\_neighbors option. The p parameter also affects the distance metric that is used to determine the neighbors. The Manhattan distance is utilized when p = 1, and the Euclidean distance is used when p = 2.

Mathematically:

* For p = 1: Manhattan Distance
* For p = 2: Euclidean Distance

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski')

These are just a few examples of how hyperparameters can shape the behavior of a machine learning model

**PYTHON PROGRAM:**

Certainly! Here's a Python program that demonstrates how to use ensemble methods (Random Forest) and hyperparameter tuning (Grid Search) using the popular scikit-learn library:

python

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

# Load a sample dataset (you can replace this with your own dataset)

data = load\_iris()

X = data.data

y = data.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Random Forest Classifier

rf\_classifier = RandomForestClassifier()

# Define a parameter grid to search through

param\_grid = {

'n\_estimators': [10, 50, 100, 200], # Number of trees in the forest

'max\_depth': [None, 10, 20, 30], # Maximum depth of the trees

'min\_samples\_split': [2, 5, 10], # Minimum samples required to split an internal node

'min\_samples\_leaf': [1, 2, 4] # Minimum number of samples required to be at a leaf node

}

# Create a GridSearchCV object to find the best hyperparameters

grid\_search = GridSearchCV(estimator=rf\_classifier, param\_grid=param\_grid, cv=5)

# Fit the grid search to the training data

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and best estimator

best\_params = grid\_search.best\_params\_

best\_estimator = grid\_search.best\_estimator\_

# Print the best parameters

print("Best Hyperparameters:")

print(best\_params)

# Evaluate the model on the test data

accuracy = best\_estimator.score(X\_test, y\_test)

print(f"Accuracy on Test Data: {accuracy:.2f}")

This code uses the Iris dataset as an example, but you can replace it with your own dataset. It first creates a Random Forest Classifier, then uses GridSearchCV to search for the best hyperparameters. Finally, it evaluates the model's accuracy on the test data. Remember to install scikit-learn (`pip install scikit-learn`) if you haven't already.

START BUILDING THE MACHINE LEARNING MODEL USING IBM CLOUD WASTON STUDIO

# **DEPLOYING MODELS WITH WASTON MACHINE LEARNING:**

You can use IBM Watson Machine Learning to deploy models, scripts, and functions, manage deployments, and prepare assets for distribution to provide predictions and insights.

**IRIS FLOWER DATASET:**

The Iris flower data set, often known as Fisher's Iris data set, is a multivariate data set that was developed and popularized by British statistician and biologist Ronald Fisher in his 1936 paper The use of many measurements in taxonomic issues as an example of linear discriminant analysis. It is frequently dubbed Anderson's Iris data set since Edgar Anderson collected the data to assess the morphologic variance of Iris blossoms of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, picked on the same day, and measured by the same person with the same apparatus."

Lets we see the example for iris flower data set

# PYTHON PROGRAM:

EXAMPLE:

**import** pandas as pd

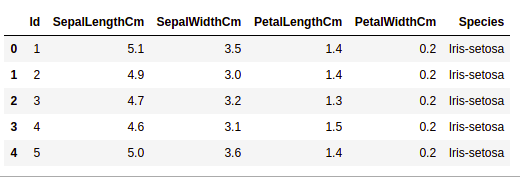
# Reading the CSV file

df **=** pd.read\_csv("Iris.csv")

# Printing top 5 rows

df.head()

OUTPUT:



**GETTING INFORMATION ABOUT THE DATASET:**

The shape parameter will be used to determine the shape of the dataset.

EXAMPLE:

df.shape

OUTPUT:

150,6

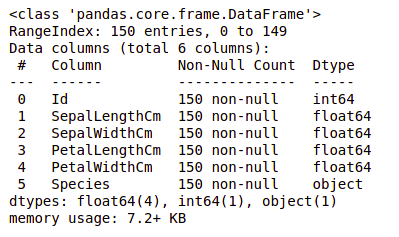
We can observe that the dataframe comprises 6 columns and 150 rows.

Now consider the columns and their data types. We shall use the info() method for this.

EXAMPLE:

df.info()

OUTPUT:

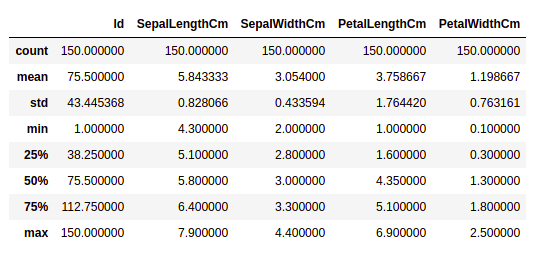


As we can see, just one column has categorical information, while the others are all numeric columns with non-Null entries.

Let's use the describe() method to acquire a quick statistical summary of the dataset. The describe() method performs fundamental statistical computations on the dataset, such as extreme values, data point count, standard deviation, and so on. Any missing or NaN value is skipped automatically. The describe() function provides an accurate representation of data distribution.

EXAMPLE:

df.describe()

OUTPUT: 

Each column's count is shown, as well as its mean, standard deviation, minimum and maximum values.

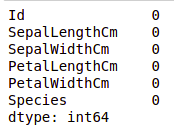
**CHECKING MISSING VALUES:**

We'll see if there are any missing values in our data. When no information is provided for one or more elements, or for the entire unit, missing values can occur. The isnull() method will be used.

EXAMPLE:

df.isnull().sum()

OUTPUT:



CHECKING DUPLICATES:

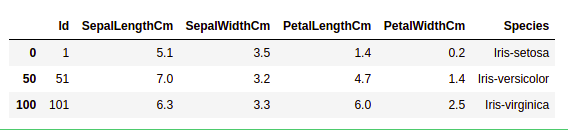
Let's see whether there are any duplicates in our dataset. The drop\_duplicates() method in Pandas assists in deleting duplicates from a data frame.

EXAMPLE:

data **=** df.drop\_duplicates(subset **=**"Species",)

data

OUTPUT:

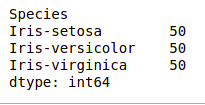


As we can see, there are just three distinct species. Let's see if the dataset is balanced, that is, if all of the species have an equal number of rows. The Series.value\_counts() function will be used. This function returns a Series with unique value counts.

EXAMPLE:

df.value\_counts("Species")

OUTPUT:



We can see that each species has an equal number of rows, hence no entries need be deleted.

HISTOGRAMS:

A Histogram is a variation of a bar chart in which data values are grouped together and put into different classes. This grouping enables you to see how frequently data in each class occur in the dataset. The histogram graphically shows the following

* Frequency of different data points in the dataset.
* Location of the center of data.
* The spread of dataset.
* Skewness/variance of dataset.
* Presence of outliers in the dataset.

Histograms allow seeing the distribution of data for various columns. It can be used for uni as well as bi-variate analysis.

EXAMPLE:

# importing packages

**import** seaborn as sns

**import** matplotlib.pyplot as plt

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

axes[0,0].set\_title("Sepal Length")

axes[0,0].hist(df['SepalLengthCm'], bins**=**7)

axes[0,1].set\_title("Sepal Width")

axes[0,1].hist(df['SepalWidthCm'], bins**=**5);

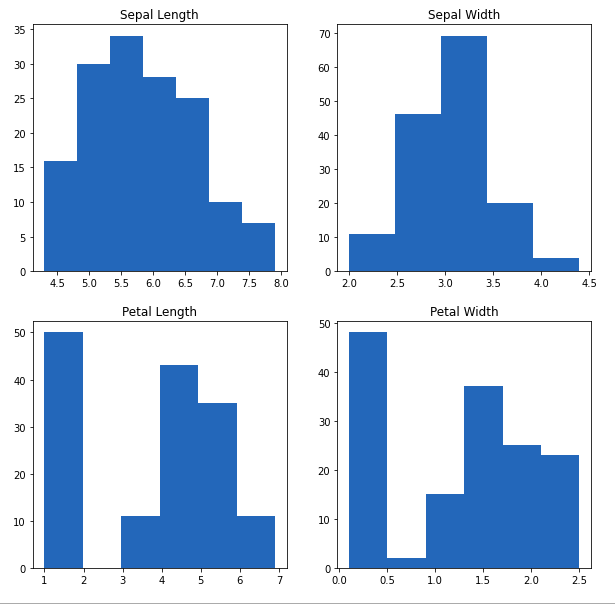
axes[1,0].set\_title("Petal Length")

axes[1,0].hist(df['PetalLengthCm'], bins**=**6);

axes[1,1].set\_title("Petal Width")

axes[1,1].hist(df['PetalWidthCm'], bins**=**6);

OUTPUT:



From the above plot, we can see that

* The largest frequency of sepal length is between 30 and 35, or 5.5 and 6
* The maximum frequency of sepal width is roughly 70, which is between 3.0 and 3.5.
* The petal length has a frequency of roughly 50, which is between 1 and 2.
* The petal width has the largest frequency between 40 and 50, which is between 0.0 and 0.5.

HISTOGRAM WITH DISPLOT PLOT:

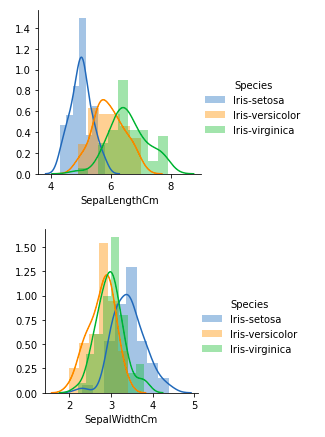
Distplot is mostly used to depict a univariant set of observations using a histogram, i.e. only one observation and hence one specific column of the dataset.

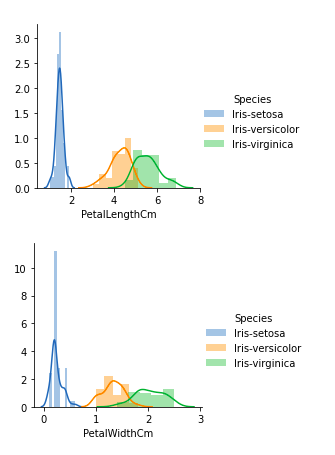
EXAMPLE:

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "SepalLengthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "SepalWidthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "PetalLengthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "PetalWidthCm").add\_legend()    plt.show() |

Lets we see the output for above program…

OUTPUT:





From the above plots, we can see that

* There is a lot of overlapping in the case of Sepal Length.
* There is also a great deal of overlapping in the case of Sepal Width.
* There is relatively little overlapping in the case of Petal Length.
* There is likewise very little overlapping in the case of Petal Width.

So we can use Petal Length and Petal Width as the classification feature.

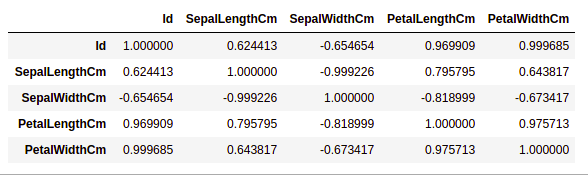
HANDLING CORELATION:

Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

EXAMPLE:

data.corr(method**=**'pearson')

OUTPUT:



HEATMAPS:

Heatmap is a way to show some sort of matrix plot. To use a heatmap the data should be in a matrix form. By matrix we mean that the index name and the column name must match in some way so that the data that we fill inside the cells are relevant.

The heatmap is a data visualization approach that uses colors in two dimensions to examine a dataset. Essentially, it demonstrates a relationship between all numerical variables in the dataset. In simplest terms, we may use the heatmaps to visualize the previously discovered association.

Lets we see the example

EXAMPLE:

# importing packages

**import** seaborn as sns

**import** matplotlib.pyplot as plt

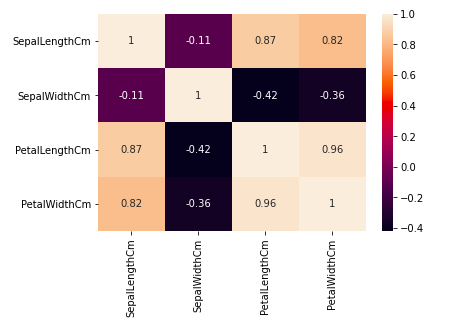
sns.heatmap(df.corr(method**=**'pearson').drop(

  ['Id'], axis**=**1).drop(['Id'], axis**=**0),

            annot **=** True);

 plt.show()

OUTPUT:



From the above graph, we can see that

* Petal breadth and length have a strong relationship.
* Petal length and sepal breadth have a strong relationship.

Petal width and Sepal length have a strong relationship

HANDLING OUTLIERS:

Outliers are data items or objects that differ dramatically from the rest of the (so-called normal)objects. Errors in measurement or execution can cause them. Outlier mining refers to the process of detecting outliers. There are numerous methods for detecting outliers, and the removal process is the same as removing a data item from the Panda's dataframe.

Consider the iris dataset and create a boxplot for the SepalWidthCm column.

EXAMPLE:

# importing packages

**import** seaborn as sns

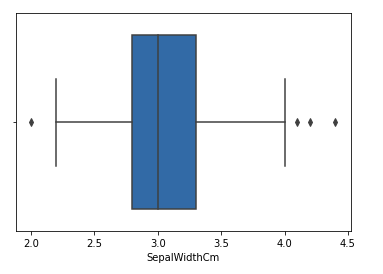
**import** matplotlib.pyplot as plt

# Load the dataset

df **=** pd.read\_csv('Iris.csv')

sns.boxplot(x**=**'SepalWidthCm', data**=**df)

OUTPUT:



In the above graph, the values above 4 and below 2 are acting as outliers.

To remove the outlier, follow the same procedure as removing an entry from the dataset using its exact position in the dataset, because the end result of all of the above methods of detecting outliers is a list of all data items that satisfy the outlier definition according to the method used.

**CONTINUE BUILDING THE PROJECT BY DEPLOYING THE MODEL AND INTEGRATING IT INTO APPLICATION**

To solve a problem by training the machine learning models, TrueFoundry helps you track different experiments and makes it easy and intuitive to deploy models with best practices and make it available for public use in a matter of minutes.

In this example, we train a model that can classify a flower of the iris genus into one of three species based on size measurements of its petal and sepal.

The iris dataset contains three different species :

* Iris Setosa
* Iris Versicolor
* Iris Virginica

TrueFoundry provides two libraries for simplifying your ML workflows:

**MLFoundry**

We shall use 5 different APIs from MLFoundry in this example. They are:

1. **log\_params** - use it to log hyper-parameters of the current experiment
2. **log\_dataset** - used to log the entire dataset
3. **log\_metrics** - log metrics like accuracy scores, f1 scores
4. **set\_tags** - add tags to your experiment for easy filtering later on
5. **log\_model** - to save a model including the trained weights

ServiceFoundry

Open an IPython notebook - you can either use Jupyter running locally on your machine or a Google Colab notebook that runs on the cloud.

CONCLUSION:

Watson Studio, Watson Machine Learning, and Cloud Object Storage are highly powerful, well-integrated, incredibly flexible, and simple-to-use tools for data scientists and AI engineers, as demonstrated in this lesson. Watson Studio provides a collaborative environment and tools for you to work on data to solve business challenges. You can select the tools you require for data analysis and visualization, data cleansing and shaping, streaming data intake, and machine learning model creation and training.