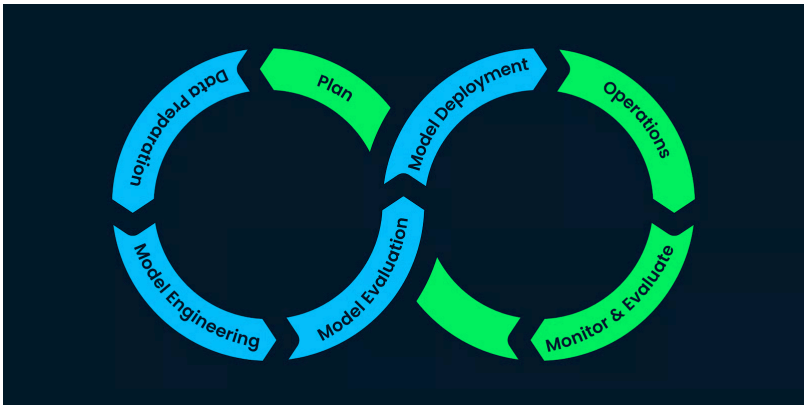
Machine Learning Lifecycle model with MLflow tracking

- Hands-on tutorial
- The scope for this tutorial is from **Plan to Model Evaluation**



Resources:

For more information refer to The Machine Learning Life Cycle Explained article.

0. Prerequisites

This sample tutorial is executed in Google Colab Notebook, a web interface which runs in a Cloud environment. Alternatively, you can also use Jupyter Notebook to run in your desktop machine. For more information refer to installation and setup instructions in Jupyter Notebook website.

- Google Colab hosts Numpy, Pandas, Scikit learn, Matplotlib and seaborn libraries. You can import these modules directly in your notebook.
- MLflow is not a pre-installed library and so, it needs to be installed as shown

```
# MLflow Installation
!pip install mlflow --quiet
```

Show hidden output

```
# Import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import neighbors
import xgboost as xgb
import mlflow
import mlflow.sklearn
```

1. Plan

Feasibility Study

1. **Availability of the data:**
    - Check for available datasets that can be used for the project and if it is trainable
2. **Applicability:**
    - Check if the dataset will help solve the problem statement
3. **Legal constraints:**
    - Check if the data and models meets legal requirements
4. **Scalabilty:**
    - Check if both model and data can be scalable
5. **Explainability:**
    - Check if the models have explainabilty
6. **Availability of resources:**
    - Check if you have computing resources

1A. Problem Statement

- **Identify the flower type with the provided flower measurments.**
- **Use Classfication model to identify the solution**

2. Data Preparation/ Exploratory Data Analysis (EDA)

- IRIS dataset imported from Scikit Learn library
- The IRIS flower dataset consists of physical measurements of sepals and petals in `cm` for the three flowers **setosa, versicolor, virginica**.
- It has a total of 150 samples with 50 samples per species.
- IRIS is stored as a multi-dimensional array or 4-dimensional array or 4D dataset.

```
# Import iris dataset
from sklearn.datasets import load_iris
iris=load_iris()
```

```
#Exploring the dataset to check which part of the data can be converted into a table
print("Column names:", iris.feature_names)
print("Rows,Columns:",iris.data.shape)
```

```
Column names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal w
Rows,Columns: (150, 4)
```

```
print("Data:",iris.data[:10]) #first 10 rows of data with the flower measurements
print("Target names/flower identifier names/metadata:", iris.target_names) #Metadata
print("Target/Flower measurements identifier", iris.target) # 0:setosa, 1:versicolor
```

Show hidden output

```
print(iris.target_names)
print(iris.target)
print(iris.target.shape) #The identifier are stored in a separate 1 dimensional arra
```

Show hidden output

Show code

Show hidden output

Show code

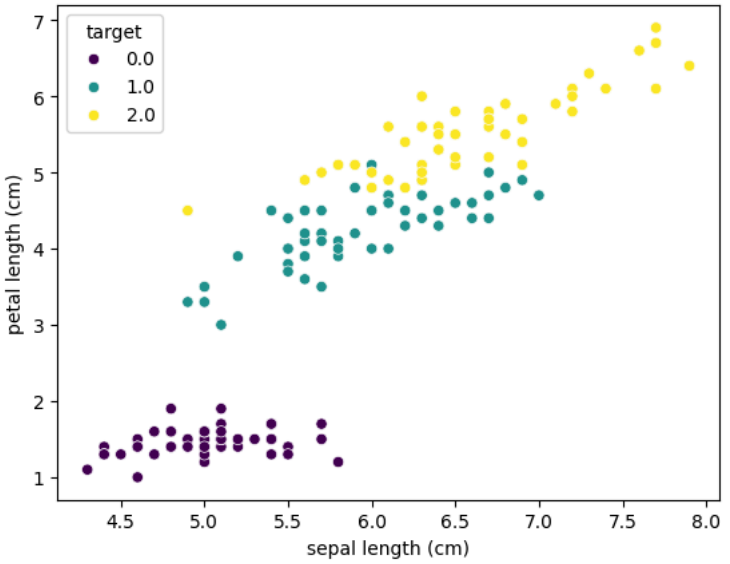
Show hidden output

```
#create dataframe with Pandas & Numpy
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                  columns= iris['feature_names'] + ['target'])
```

```
df.sample(10) # a random sample of 10 rows in the dataframe
```

Show hidden output

```
#Visualizing the data to identify the flower types based on sepal length and petal l
sns.scatterplot(data=df, x=df['sepal length (cm)'], y=df['petal length (cm)'], hue='
#plt.legend(labels=['setosa' 'versicolor' 'virginica'], title='Iris Species')
plt.show()
```



3. Model Engineering

- Train and Test the dataset
- A k-NN (k Nearest neighbor) classification model is used for identifying the flower type
- k-NN model uses the Euclidean distance(straight line) concept and similarity(Minkowski Distance) between existing data point and the new data point to classify or predict the flower type(target)

```
# Preparing the data for training
X= df.drop('target', axis=1)
X
```

Show hidden output

Next steps: Generate code with X | New interactive sheet

```
y=df['target']
y
```

Show hidden output

```
# Split the dataset into 80% training and 20% testing
# random_state ensure the same or identical data is used from the dataset each time
# stratify is used to prevent bad random splitting. it helps select equal number of
# it does a train, test split
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,st
```

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

```
(120, 4) (30, 4) (120,) (30,)
```

MLflow

Mlflow is a tool for managing Machine Learning Lifecycle model. It tracks model iterations as experiments, manages your ML models, tracks models parameters and provides visualized metrics. For detailed information refer to the MLflow documentation.

MLflow can be integrated in your models in two ways:

1. **MLflow UI**
2. **Implementing MLflow directly in your code**

The example below shows you how to incorporate MLflow in your code.

Step 1: Installation

Use the `pip` command to install mlfow and import the mlflow library

MLflow Installation

!pip install mlflow --quiet
import mlflow

import mlflow.sklearn

Step 2. **MLflow** instantiation

- Instantiate MLflow to start model execution(training) tracking. The below code starts an instance for tracking the ML model.
- You can list all the parameters that you need to track using the `log_param` function. The example ML model used is k-NN(K nearest neighbors)
- The model parameter that is tracked in the `k-value` which is defined as `k_neighbors`

```
#Setting hyper parameter for k-NN mode. k-NN model looks for 4 closest data points(r
# classification of the new point
n_neighbors=20
with mlflow.start_run():
  mlflow.log_param('k_neighbors',n_neighbors)
```

Step 3. Start logging

- The code below starts recording the model trainings results such as parameters, artifacts and metrics
- Each training cycle is logged in as experiments and has a specific ID

```
mlflow.sklearn.autolog()
```

Step 4. Load the model

- The kNN Classifier model is loaded. kNN is a classification model which identifies and categorizes data in the respective class.
  - IRIS is Scikit learn's multiclass flower dataset used for Classification problems in machine learning.

```
knn=neighbors.KNeighborsClassifier(n_neighbors)
knn
```

```
▾  KNeighborsClassifier
KNeighborsClassifier(n_neighbors=20)
```

Step 5. Train the model

- The model is trained using the IRIS dataset
- You should define the list of parameters required for the model in order execute the ML model. In the below code, the kNN model has 1 mandatory parameter `k_neighbors`
- The `k_neighbors` parameter can be tuned to improve training score and accuracy metrics score
- Each time the parameter is tuned `mlflow` logs this information

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

```
2026/02/05 07:09:17 INFO mlflow.utils.autologging_utils: Created MLflow autologging r
2026/02/05 07:09:17 WARNING mlflow.utils.autologging_utils: MLflow autologging encour
```

KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=20)
```

Step 6. Predict the label or class

- The kNN model does an approximate guess for classifying the flower species

```
y_predict=knn.predict(X_test)
y_predict
```

```
array([0., 2., 1., 1., 0., 1., 0., 0., 2., 1., 2., 2., 2., 1., 0., 0., 0.,
       1., 1., 1., 0., 2., 1., 2., 2., 2., 1., 0., 2., 0.])
```

Model Evaluation

- This phase checks how the model has performed and how accurate are the predictions.
- MLflow has built-in evaluation metrics such as accuracy, loss and many more.
- You can also use the MLflow UI for graphical visualization of parameters, artifacts and metrics. Refer to the installation guide for MLflow UI

```
# manual accuracy score calculation
accuracy=knn.score(X_test,y_test)
accuracy
```

```
0.9333333333333333
```

Step 7: Logging metrics

- You can also manually log your required metrics using the below code

```
mlflow.log_metric("accuracy",accuracy)
```

Step 8: Retrive log data

- All the logs are stored in a temp file and needs to be retreived
- The `search_runs` function retreives the log which contains all the tracked experiments, parameters, artificats and metrics

```
runs=mlflow.search_runs()
```

Step 8: View the log

- You can use pandas dataframe to structure, organize and manipulate the data in a table format.

```
knn_log=pd.DataFrame(data=runs)
knn_log
```

Show hidden output

Step 8A: View your required metrics

- The example below extracts only the k-value, training metrics score and accuracy score
- Based on the results you can determine the best model that outputs highest accuracy in the class predictions
- The model is overfitting. The reason is that only one training dataset was used against the different k value. As the k-value gets higher the accuracy is going higher. In short, the model memorized the training dataset too well and can't handle new data.
- To determine if the model is performing well you should evaluate using different training datasets and multiple k-values

```
knn_log[['params.n_neighbors','metrics.training_score','metrics.training_accuracy_sc
```

	params.n_neighbors	metrics.training_score	metrics.training_accuracy_score
0	None	NaN	NaN
1	20	0.983333	0.983333
2	None	NaN	NaN
3	None	NaN	NaN
4	7	0.975000	0.975000
5	None	NaN	NaN
6	None	NaN	NaN
7	8	0.975000	0.975000
8	None	NaN	NaN
9	None	NaN	NaN
10	5	0.966667	0.966667
11	None	NaN	NaN
12	None	NaN	NaN
13	3	0.958333	0.958333
14	None	NaN	NaN
15	None	NaN	NaN
16	None	NaN	NaN
17	None	NaN	NaN
18	4	0.958333	0.958333
19	None	NaN	NaN

Step 9: Stop the mlflow instance

- Use the `end_run` function to terminate the instance

```
mlflow.end_run()
```

xGBoost model

↳ 10 cells hidden