**Machine learning with IBM Watson**

**Phase 2: problem definition and Design approach**

**Problem Definition:**

The project involves training a machine learning model using IBM cloud Watson studio and deploying it as a webservice. The goal is to become proficient in predictive analytics by creating model that can predict outcomes in real- time. The project encompasses defining the predictive use case, selecting a suitable dataset, training a machine learning model, deploying the model as a web service, and integrating it into applications**.**

**Design Approach:**

**1. Define the Problem:**

Clearly define the problem you want to solve. Understand the business objectives and how machine learning can help achieve them.

**2. Data Collection:**

Gather relevant data for your problem. This could involve scraping websites, using APIs, accessing databases, or generating synthetic data.

**3. Data Preprocessing:**

Clean and preprocess the data. This includes handling missing values, outlier detection, and data normalization.

**4. Exploratory Data Analysis (EDA):**

Conduct EDA to gain insights into the data. Visualize the data, perform statistical analysis, and identify patterns.

**5. Feature Engineering:**

Create relevant features that will help the model understand the problem better. This can involve transforming existing features or creating new ones.

**6. Data Splitting:**

Divide the data into training, validation, and test sets. The training set is used to train the model, the validation set to fine-tune hyperparameters, and the test set to evaluate the model's performance.

**7. Model Selection:**

Choose an appropriate machine learning algorithm or framework for your problem. This could be a decision tree, random forest, neural network, or any other suitable model**.**

**8. Model Training:**

Train the selected model on the training data. Adjust hyperparameters to improve performance as needed.

**9. Model Evaluation:**

Evaluate the model's performance using appropriate metrics (e.g., accuracy, precision, recall, F1 score, RMSE, etc.) on the validation set. Make sure to avoid overfitting**.**

**10. Hyperparameter Tuning:**

Fine-tune the model by adjusting hyperparameters. You can use techniques like grid search, random search, or Bayesian optimization.

Deployment: once you are satisfied with the performance and response quality of your prediction system, incorporate it into your application, website, or platform.

# Import necessary libraries

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load the dataset (replace 'iris.csv' with your data)

data = pd.read\_csv('iris.csv')

# Data preprocessing

X = data.drop('target\_column\_name', axis=1) # Features

y = data['target\_column\_name'] # Target variable

# Split the data 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)

# Standardize features (if needed)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create a machine learning model (Logistic Regression in this case)

model = LogisticRegression()

# Train the model

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print("Accuracy: ", accuracy)

print("Confusion Matrix:\n", confusion)

print("Classification Report:\n", report)