# MACHINE LEARNING MODEL DEPLOYMENT WITH IBM CLOUD WATSON STUDIO [PHASE-4]

## Introduction:

Deploying a trained machine learning model as a web service and integrating it into applications is a crucial step in turning your machine learning project into a real-world, user-facing solution. In this guide, we'll walk you through the process of deploying your model as a web service on IBM Cloud Watson Studio and integrating it into applications using the provided API endpoint.

## 1. Prepare Your Model for Deployment

Before deploying your model, ensure that it is properly trained, tested, and ready for production use. You should have a saved and serialized version of your model.

## 2.Set Up IBM Cloud Watson Studio

- If you don't already have an IBM Cloud account, sign up for one.
- Access IBM Cloud Watson Studio and create a project that will host your deployed model.

## 3. Deploy Your Model in Watson Studio

- Inside your project, create a deployment space.
- Choose your model and deploy it to the deployment space. Watson Studio offers tools for deploying various types of models, including machine learning and deep learning model

## 4. Configure Your Deployment

- Define the runtime environment and resources for your deployed model. You can choose CPU or GPU configurations based on your model's requirements.
- Set up authentication and access controls to secure your deployed model.

## 5. Access the API Endpoint

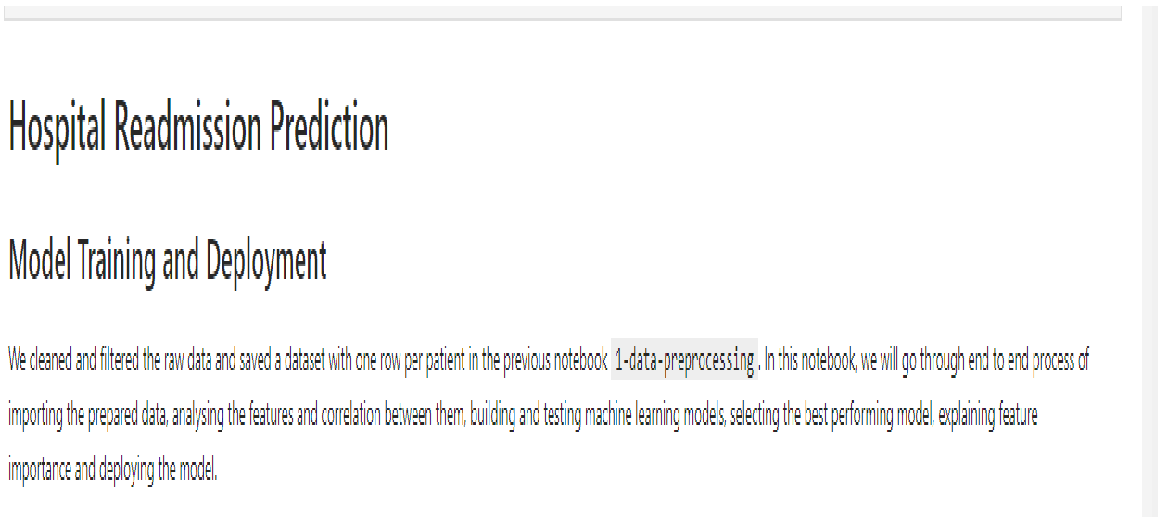
Once your model is deployed, you will receive an API endpoint URL. This URL will allow you to send requests to your model and receive predictions.

## 6.Integrate the Model into Applications

Now that your model is deployed and has an accessible API endpoint, you can start integrating it into your applications.

- Web Applications: If you are developing a web application, you can make HTTP requests to the API endpoint using libraries like Axios, fetch, or Python's requests module. Parse the responses to display predictions or take further actions in your application.
- Mobile Applications: For mobile apps, you can use libraries or SDKs provided by the platform (e.g., iOS or Android) to send requests to the API endpoint and process the model's responses.
- IoT Device: If you are deploying your model to IoT devices, you can use MQTT or HTTP protocols to communicate with the API endpoint.

## Step1:



Hospital Readmission Prediction

Model Training and Deployment

We cleaned and filtered the raw data and saved a dataset with one row per patient in the previous notebook `1-data-preprocessing`. In this notebook, we will go through end to end process of importing the prepared data, analysing the features and correlation between them, building and testing machine learning models, selecting the best performing model, explaining feature importance and deploying the model.

```
In [3]: import pandas as pd
        pd.set_option('display.max_columns', None)

        import matplotlib.pyplot as plt
        import matplotlib.patches as mpatches
        import numpy as np
        import os

        import seaborn as sns
        sns.set(style='darkgrid',palette="deep")
        from zipfile import ZipFile
        import joblib
        import string
        import random


        from sklearn.model_selection import train_test_split

        from sklearn.compose import ColumnTransformer
        from sklearn.pipeline import Pipeline
        from sklearn.impute import SimpleImputer
        from sklearn.preprocessing import StandardScaler, OneHotEncoder

        from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

        from sklearn import metrics
```

## Load training data

We load the training data prepared in the notebook `1-data-preprocessing`. Training data set contains 1 row per patient with patients health records, admission and discharge records, lab, diagnoses and procedures details and the target variable to indicate whether the patient was readmitted within 30 days of discharge or not.

```
In [4]: hospital_details = project.get_file("training-data.csv")
        hospital_details.seek(0)
        df_hospital = pd.read_csv(hospital_details)
        df_hospital.head()
```

```
5]: def plot_feature_importance(feature_list, feature_importances, title='Feature Importance Plot'):
        """
        Function to Plot Feature Importances
        """
        features = feature_list
        importances = feature_importances
        indices = np.argsort(importances)[-10:]

        plt.figure(figsize=(12,7))
        plt.title(title, fontsize=16, fontweight='bold')
        plt.barh(range(len(indices)), importances[indices], color='b', align='center')
        plt.yticks(range(len(indices)), [features[i] for i in indices])
        plt.xlabel('Relative Importance')

        return
```

```
In [7]: numerical_cols=['TIME_IN_HOSPITAL','NUM_LAB_PROCEDURES','NUM_PROCEDURES','NUM_MEDICATIONS','NUMBER_OUTPATIENT','NUMBER_EMERGENCY','NUMBER_DIAGNOSES']


        target_col='READMITTED'
        categorical_cols=['RACE','GENDER','AGE','WEIGHT','ADMISSION_TYPE_ID','DISCHARGE_DISPOSITION_ID','ADMISSION_SOURCE_ID','PAYER_CODE',
                          'DIAG_1','DIAG_2','DIAG_3','MAX_GLU_SERUM','A1CRESULT','METFORMIN','REPAGLINIDE','NATEGLINIDE','CHLORPROPAMIDE','GLIMEPIRIDE','ACETOHEXAMIDE','GLIPIZIDE','(
                          'PIOGLITAZONE','ROSIGLITAZONE','ACARBOSE','MIGLITOL','TROGLITAZONE','TOLAZAMIDE','EXAMIDE','CITOGLIPTON','INSULIN','GLYBURIDE-METFORMIN','GLIPIZIDE-METFORMI
                          'METFORMIN-ROSIGLITAZONE','METFORMIN-PIOGLITAZONE','CHANGE','DIABETESMED']


        cat_pct = 0.05
        cor_pct = 0.90
```

```
In [8]: df_prep=df_hospital[numerical_cols+[target_col]+categorical_cols].copy()
```

## Visualise Input Data

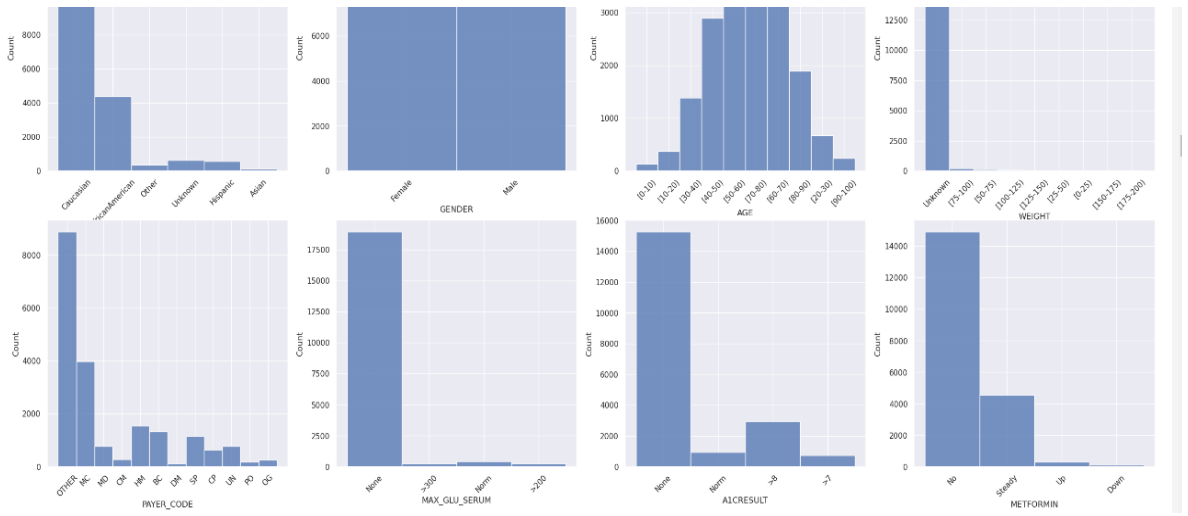
### Plot Categorical Columns

The below cell plots some of the categorical columns to understand their distribution.

```
f = plt.figure(figsize=(30,150))
c=0
for i in categorical_cols:
    if ("DIAG" not in i) & ("_ID" not in i):

        f.add_subplot(20, 4, c+1)
        plt.xticks(rotation=45)
        sns.histplot(df_prep[i])
        c=c+1
plt.show()
```

# Step2:



# Step3:

## Deploy the Model Pipeline

Deploy the saved model. The deployment name is specified in the user inputs cell above. Again, a default tag for the deployment is specified in the deployment metadata.

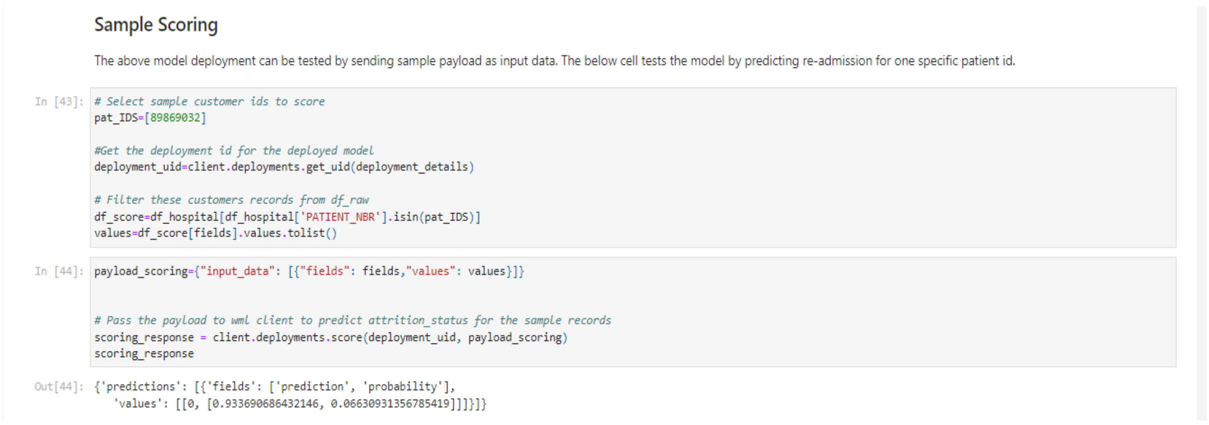
```
# deploy the model pipeline
meta_props = {
    client.deployments.ConfigurationMetaNames.NAME: deployment_name,
    client.deployments.ConfigurationMetaNames.TAGS : [ 'hospital_readmission_pipeline_deployment_tag'],
    client.deployments.ConfigurationMetaNames.SERVING_NAME: model_name.lower().replace(' ','')[:30]+''.join(random.choice(string.ascii_lowercase + string.digits) for _ in rar
    client.deployments.ConfigurationMetaNames.CUSTOM: metadata_dict
}

# deploy the model

model_uid = stored_model_details["metadata"]["id"]
deployment_details = client.deployments.create(artifact_uid=model_uid, meta_props=meta_props)
```

```
#######################################################################################

Synchronous deployment creation for uid: '5264f1c6-4f6f-4f2e-9ecc-926ca53b5898' started
```



### Sample Scoring

The above model deployment can be tested by sending sample payload as input data. The below cell tests the model by predicting re-admission for one specific patient id.

```
In [43]: # Select sample customer ids to score
         pat_IDS=[89869032]

         #Get the deployment id for the deployed model
         deployment_uid=client.deployments.get_uid(deployment_details)

         # Filter these customers records from df_raw
         df_score=df_hospital[df_hospital['PATIENT_NBR'].isin(pat_IDS)]
         values=df_score[fields].values.tolist()

In [44]: payload_scoring={"input_data": [{"fields": fields,"values": values}]}


         # Pass the payload to wml client to predict attrition_status for the sample records
         scoring_response = client.deployments.score(deployment_uid, payload_scoring)
         scoring_response

Out[44]: {'predictions': [{'fields': ['prediction', 'probability'],
            'values': [[0, [0.933690686432146, 0.06630931356785419]]]}]}
```

## Conclusion:

Deploying a machine learning model as a web service in IBM Cloud Watson Studio and integrating it into applications can be a transformative step for your project. This process allows you to bring the power of machine learning to end users and make data-driven predictions in real-world scenarios. With the API endpoint, your model becomes accessible to a wide range of applications, making it a valuable asset for your organization and its stakeholders. Remember to monitor and maintain your deployed model to ensure it continues to deliver accurate results as new data becomes available.

PRESENTED BY

1.B.Jayasudha

2.M.Sakthi

3.M.Swetha

4.C.Vinothini