Task 3 P: Design and Deploy Model using Azure Machine Learning

Part 1:

What is the Azure Machine Learning SDK for Python?

Azure Machine Learning Python SDK is a library for Python that allows users to build, train, deploy and manage machine learning models on Microsoft Azure. It provides different packages and APIs for working with Azure Machine Learning services. The SDK allows users to:

- Build, train and deploy machine learning and deep learning models
- Able to run discrete ML activity as a job that can be run locally or over the cloud
- Provides scalability and performance optimizations for training and deploying ML models
- Provides ease of monitoring and managing model performance, versions and more.

What is Azure Machine Learning Workspace?

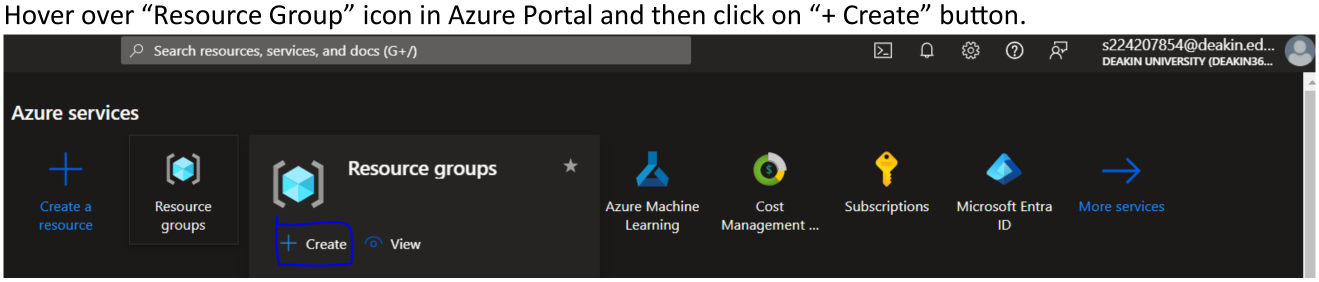
Azure Machine Learning workspace is a central place to manage resources and assets for training and managing models. The workspace keeps a history of all jobs, including logs, metrics, output and a snapshot of your scripts. It also stores references like data, models, environments and more. In workspace we can perform different tasks like:

- Jobs creation for training runs, which used to build models.
- Helps in managing data used for model training and pipelines.

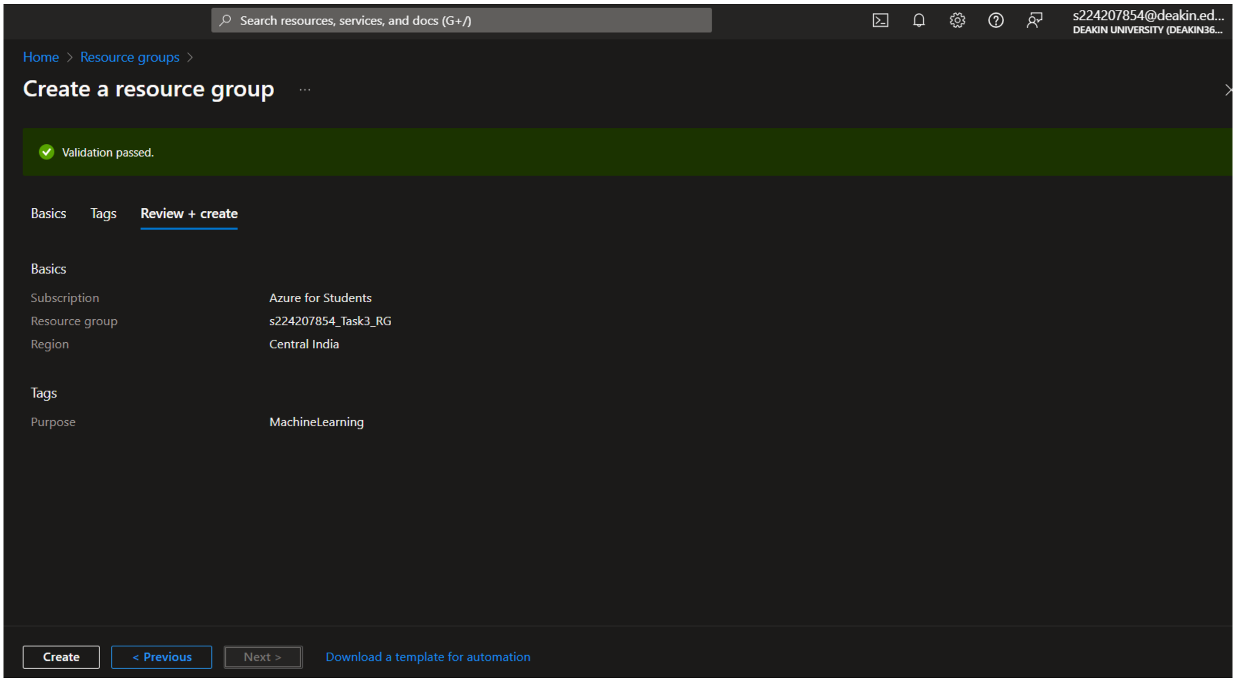
How to create a workspace for machine learning using Azure Portal?

Steps to create Azure ML workspace using Azure Portal:

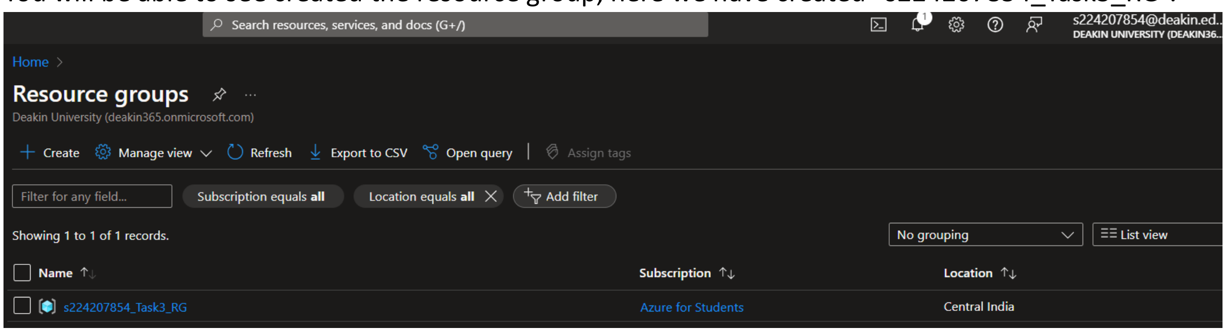
1. Sign in to Azure portal with your Deakin creds. Link -> portal.azure.com
2. Create a new resource group, using below steps:
    a. Hover over "Resource Group" icon in Azure Portal and then click on "+ Create" button.



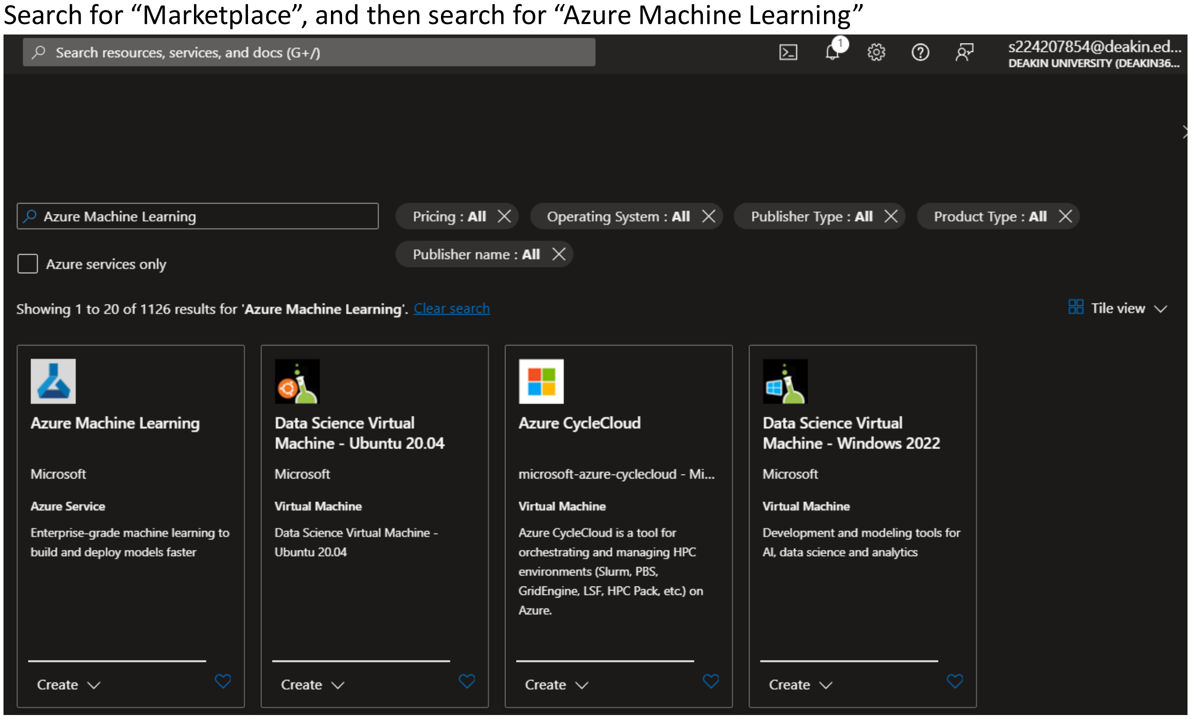
    b. Fill the required information for resource creation and before creating, please review, will be looking like below, after which click on "Create" button.



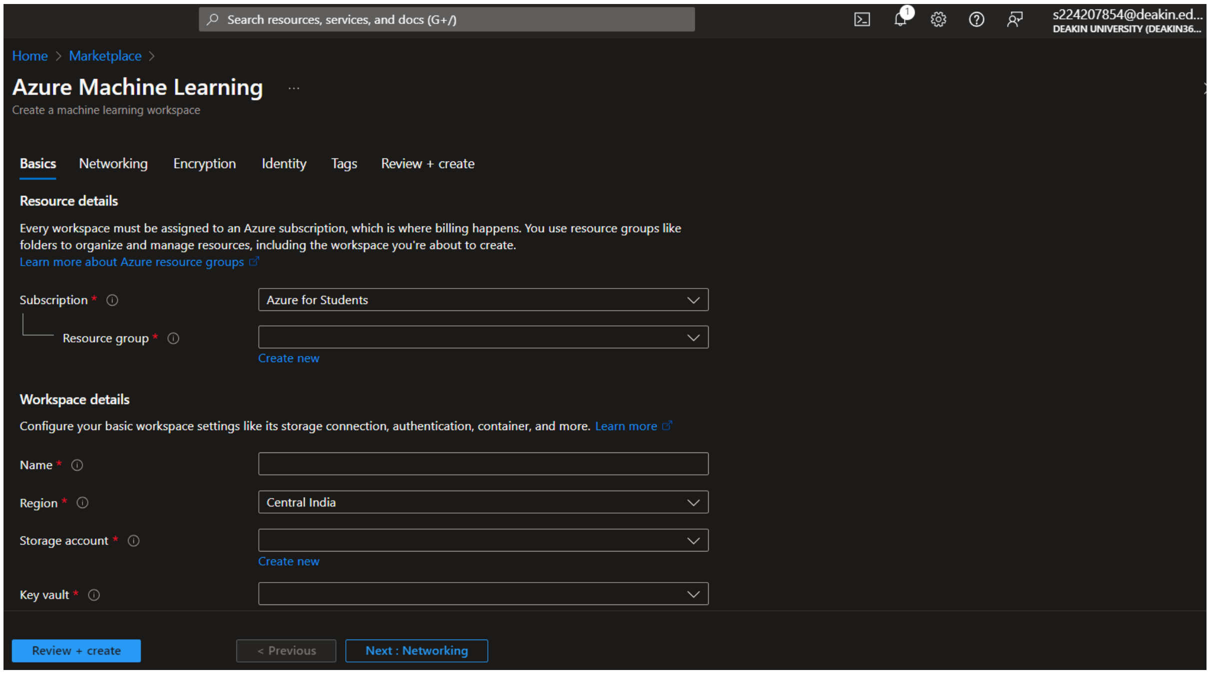
c. You will be able to see created the resource group, here we have created “s224207854_Task3_RG”.



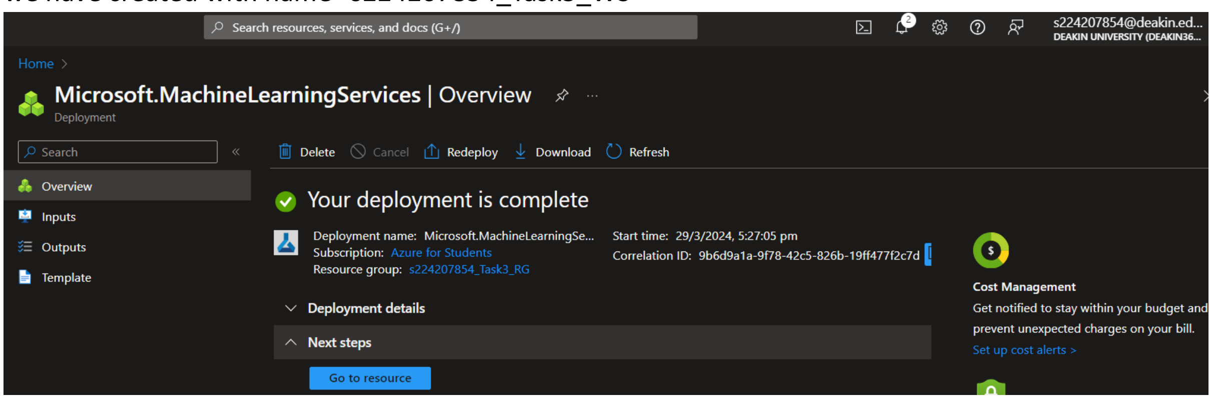
3. Search for “Marketplace”, and then search for “Azure Machine Learning”



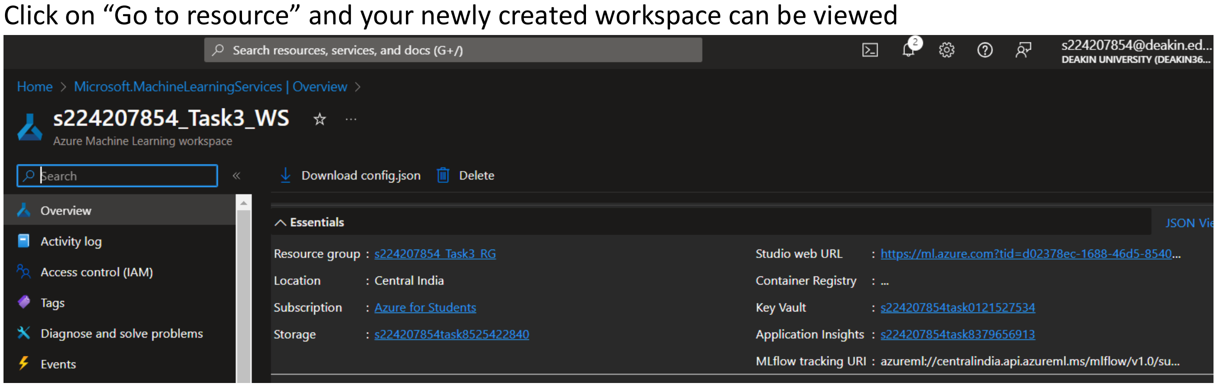
4. Click on “Create” button in Azure Machine Learning, which will redirect to this page.



5. Once required details has been filled, please go for “Review + create” button and “Create” the workspace, here we have created with name “s224207854_Task3_WS”



6. Click on “Go to resource” and your newly created workspace can be viewed

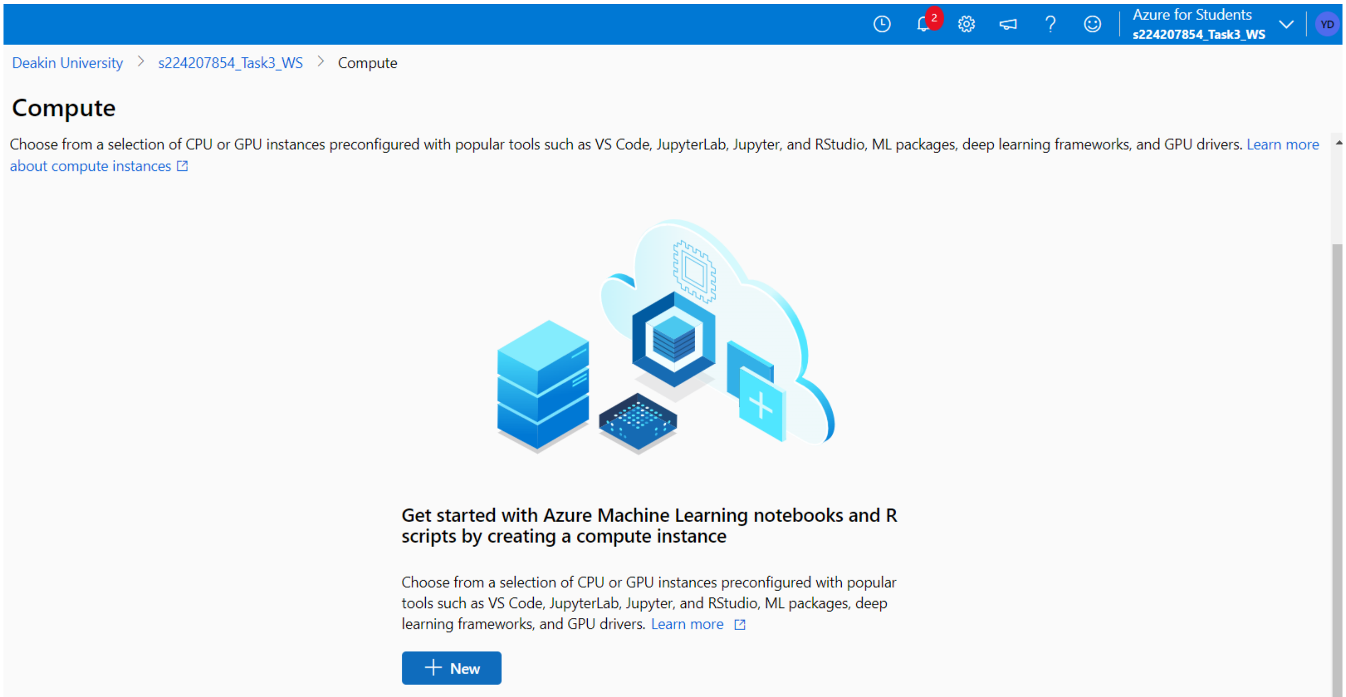


What is compute instance and how to create compute instance using Azure Portal?

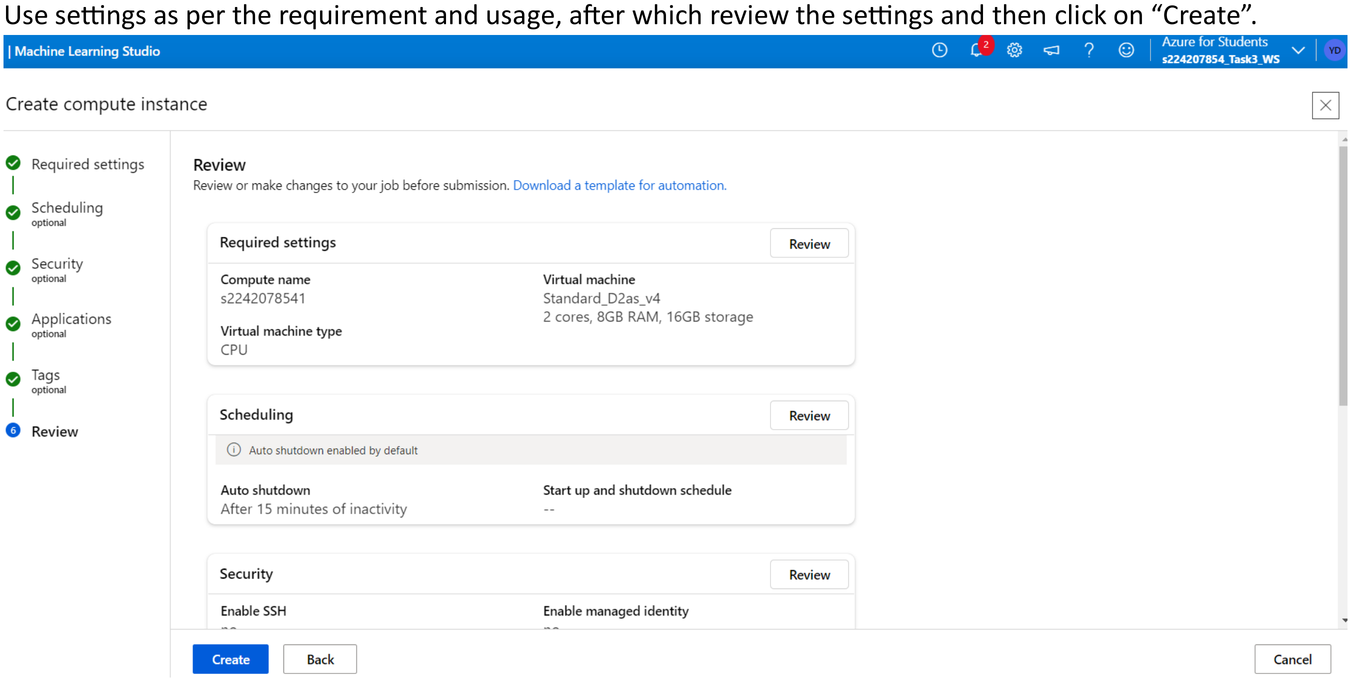
Compute Instance in Azure Machine Leaning is a virtual machine which provides a development environment for users and we can start running sample notebooks without needing to set up anything.

Step to create compute instance:

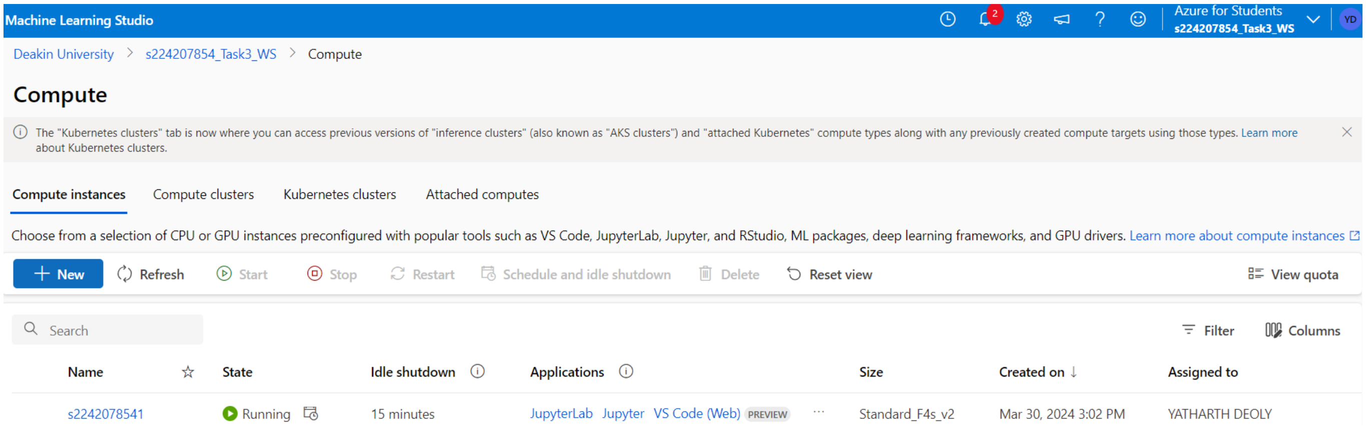
- Open your workspace and click on "Compute" with "Compute instances" as tab. Click on "+ New" button.



- Use settings as per the requirement and usage, after which review the settings and then click on "Create".



- Once deployment is completed we can check our instance status, here our instance name is “s2242078541”.



Part 2:

Introduction

In this case study we will be using python knowledge with Azure Machine Learning to train and deploy the model. Dataset that will be used here is of "***Chronic Kidney Disease***". 400 records have been provided in this dataset.

This dataset contains 25 columns, i.e.,

- **Age**: This column contains numerical data and units is in years with missing values. ColumnName: 'age'.
- **Blood Pressure**: This column contains numerical data and unit is in mm/Hg with missing values. ColumnName: 'bp'.
- **Specific Gravity**: This column contains categorical data with unique values as: (1.005,1.010,1.015,1.020,1.025) and some missing values. ColumnName: 'sg'.
- **Albumin**: This column contains categorical data with unique values as: (0,1,2,3,4,5) and some missing values. ColumnName: 'al'.
- **Sugar**: This column contains categorical data with unique values as: (0,1,2,3,4,5) and some missing values. ColumnName: 'su'.
- **Red Blood Cells**: This column contains categorical data with unique values as: (normal, abnormal) and some missing values. ColumnName: 'rbc'.
- **Pus Cell**: This column contains categorical data with unique values as: (normal, abnormal) and some missing values. ColumnName: 'pc'.
- **Pus Cell clumps**: This column contains categorical data with unique values as: (present, notpresent) and some missing values. ColumnName: 'pcc'.
- **Bacteria**: This column contains categorical data with unique values as: (present, notpresent) and some missing values. ColumnName: 'ba'.
- **Blood Glucose Random**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'bgr'.
- **Blood Urea**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'bu'.
- **Serum Creatinine**: This column contains numerical data and units is in mgs/dl with missing values. ColumnName: 'sc'.
- **Sodium**: This column contains numerical data and units is in mEq/L with missing values. ColumnName: 'sod'.
- **Potassium**: This column contains numerical data and units is in mEq/L with missing values. ColumnName: 'pot'.
- **Hemoglobin**: This column contains numerical data and units is in gms with missing values. ColumnName: 'hemo'.
- **Packed Cell Volume**: This column contains numerical data with missing values. ColumnName: 'pcv'.
- **White Blood Cell Count**: This column contains numerical data and units is in cells/cmm with missing values. ColumnName: 'wbcc'.
- **Red Blood Cell Count**: This column contains numerical data and units is in millions/cmm with missing values. ColumnName: 'rbcc'.
- **Hypertension**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'htn'.
- **Diabetes Mellitus**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'dm'.
- **Coronary Artery Disease**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: 'cad'.
- **Appetite**: This column contains categorical data with unique values as: (good, poor) and some missing values. ColumnName: 'appet'.

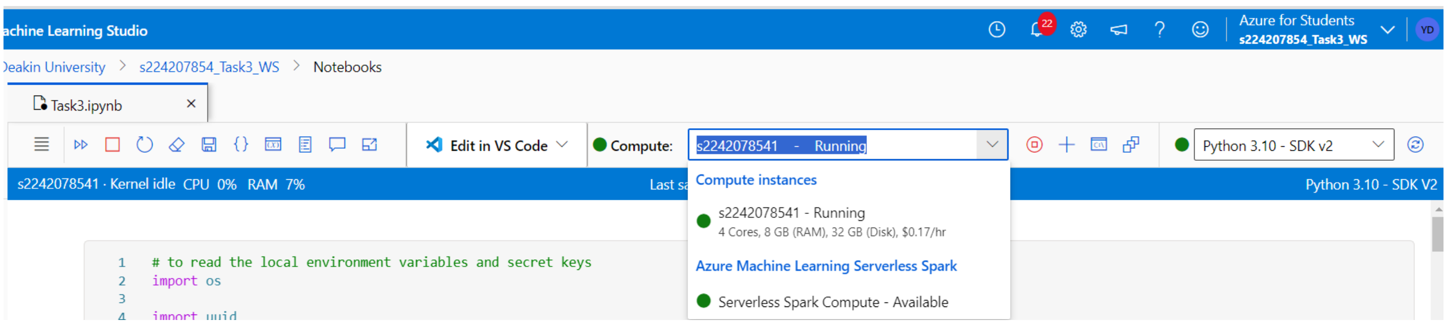
- **Pedal Edema**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: ‘pe’.
- **Anemia**: This column contains categorical data with unique values as: (yes, no) and some missing values. ColumnName: ‘ane’.
- **Class**: This column contains categorical data with unique values as: (ckd, notckd) and some missing values. ColumnName: ‘class’.

Brief information about dataset:

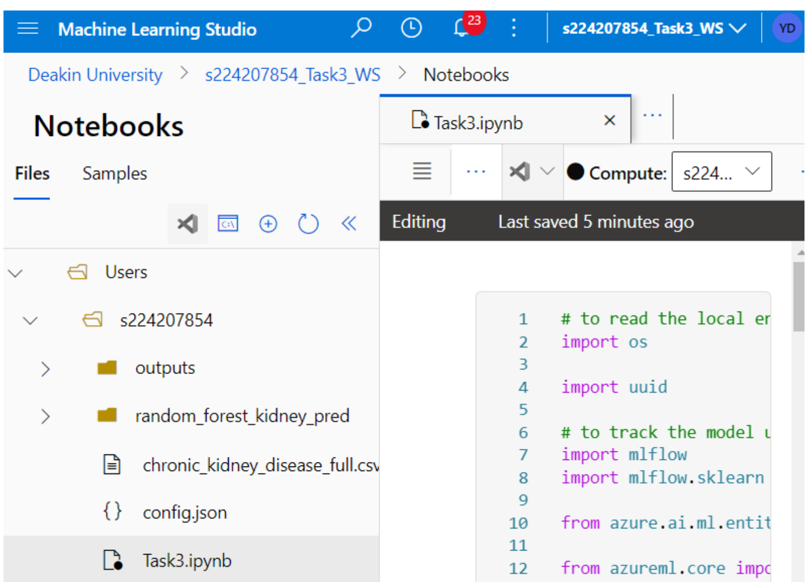
- Dataset contains missing values which has been denoted as ‘?’.
- Dataset have 24 features and 1 target class. These categories have been divided into as 11 numerical and 14 nominal.

Azure Notebook

To do this task we will be using Azure ML notebook, which can be created from “Notebooks” section. Once notebook is created, we need to have a compute instance for our notebook and setting up our kernel as “Python 3.10 – SDK v2”



In notebook section, we can upload different resources that will be used in our case study.



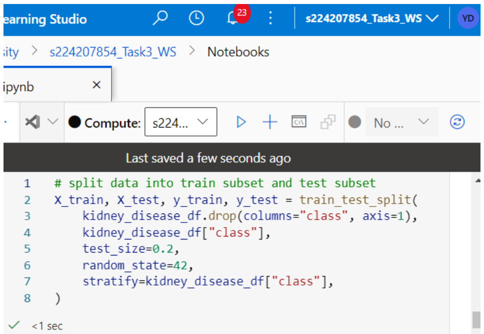
Data Loading and Data Pre-processing

- Data has been cleansed and ready for future analytics and modelling use, by using different ML techniques.
- Different steps followed here are:
    - Extracting data into pandas Dataframe.
    - Rectifying column names, as col names contains quotes, which needs to be removed.
    - Replacing missing values i.e. '?' with NAN.
    - As we have 14 categorical data that needs to be encoded, so for this we will be using 'LabelEncoder' technique from 'sklearn' package. This is a technique that converts categorical variables into numerical values.
    - As we have ample missing data in every column, so for that we will be using 'KNNImputer' technique from 'sklearn' package. This is a scikit-learn class that uses the K-Nearest Neighbors (KNN) algorithm to predict or fill in missing values in a dataset. It's a multivariate technique that considers multiple features in the dataset to estimate the missing values.
    - As we have 11 non categorical data whose values ranges a lot, so to normalize we will be using 'StandardScaler' technique from 'sklearn' package. It normalizes features by removing the mean and scaling them to unit variance.

```
kidney_disease_df = pd.read_csv("chronic_kidney_disease_full.csv", usecols=range(1, 26))
# Columns with nominal data
categorical_cols = [···

# Columns with numerical data
non_categorical_cols = [···


def encode(data):
    """function to encode non-null data"""

    data_no_null = np.array(data.dropna())  # retains only non-null values

    encoded_data = label_encoder.fit_transform(data_no_null)  # encode date

    data.loc[data.notnull()] = np.squeeze(
        encoded_data
    )  # Assign back encoded values to non-null values

    return data


def impute(data, col):
    """function to impute null data"""

    result = knn_imputer.fit_transform(data)

    if col in categorical_cols:
        return result.astype(int)

    return np.round(result, 2)


# Rectifying column names
kidney_disease_df.columns = kidney_disease_df.columns.str.replace("'", "")

# Replacing missing values i.e. '?' with NAN
kidney_disease_df = kidney_disease_df.replace("?", np.NaN)


kidney_disease_df[categorical_cols] = kidney_disease_df[categorical_cols].apply(encode)

kidney_disease_df[categorical_cols] = kidney_disease_df[categorical_cols].astype(
    "category"
)


for col in kidney_disease_df.columns:

    kidney_disease_df[[col]] = impute(kidney_disease_df[[col]], col)


# Fit and transform the scaler to the training data
kidney_disease_df[non_categorical_cols] = scaler.fit_transform(
    kidney_disease_df[non_categorical_cols]
)


kidney_disease_df.head()
```

Data Split

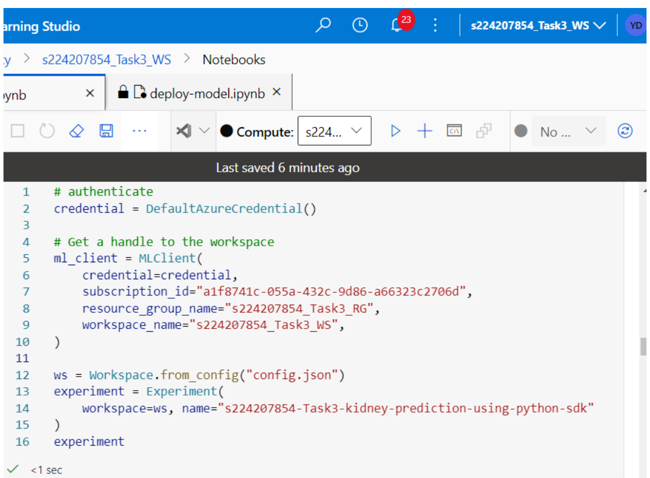
Now we will be splitting our dataframe into 80% training dataset and 20% as testing dataset. To achieve this, we will be using `train_test_split' from 'sklearn' package. This is a function in the scikit-learn library that splits a dataset into two sets: a training set and a test set. The training set is used to fit a machine learning model, while the test set is used to evaluate the model's performance.



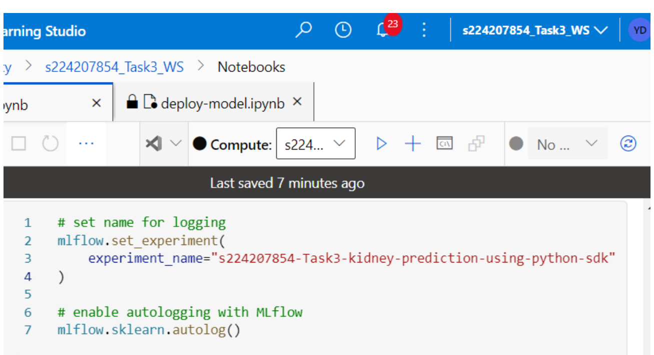
```
# split data into train subset and test subset
X_train, X_test, y_train, y_test = train_test_split(
    kidney_disease_df.drop(columns="class", axis=1),
    kidney_disease_df["class"],
    test_size=0.2,
    random_state=42,
    stratify=kidney_disease_df["class"],
)
```

Prerequisites for Azure ML

Now will be declaring MLClient which will help in managing resources, jobs and ml-flow. Once workspace is created will create an experiment, which is another foundational cloud resource that represents a collection of trials (individual model runs). Once experiment is set, will be enabling auto logging with MLflow.



```
# authenticate
credential = DefaultAzureCredential()

# Get a handle to the workspace
ml_client = MLClient(
    credential=credential,
    subscription_id="a1f8741c-055a-432c-9d86-a66323c2706d",
    resource_group_name="s224207854_Task3_RG",
    workspace_name="s224207854_Task3_WS",
)

ws = Workspace.from_config("config.json")
experiment = Experiment(
    workspace=ws, name="s224207854-Task3-kidney-prediction-using-python-sdk"
)
experiment
```



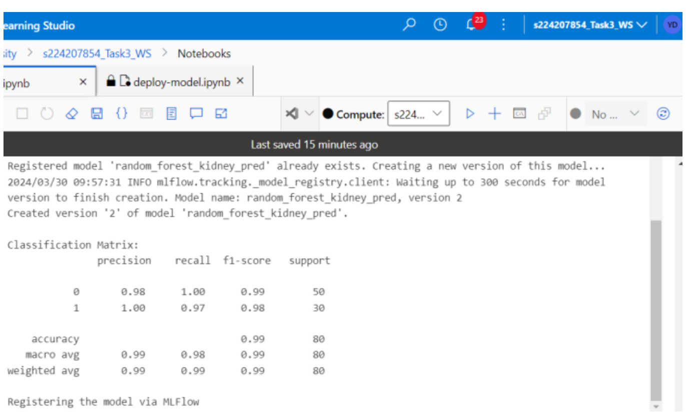
```
# set name for logging
mlflow.set_experiment(
    experiment_name="s224207854-Task3-kidney-prediction-using-python-sdk"
)

# enable autologging with MLflow
mlflow.sklearn.autolog()
```

Registering and creation of model

As our best model, will be setting up RandomForestClassifier with different set of hyperparameters. Using sklearn and mlflow libraries, will be running our ML model and will be showing report of our model. Once model is ready, will register that model to our Azure ML portal. Model will be registered using MLflow.

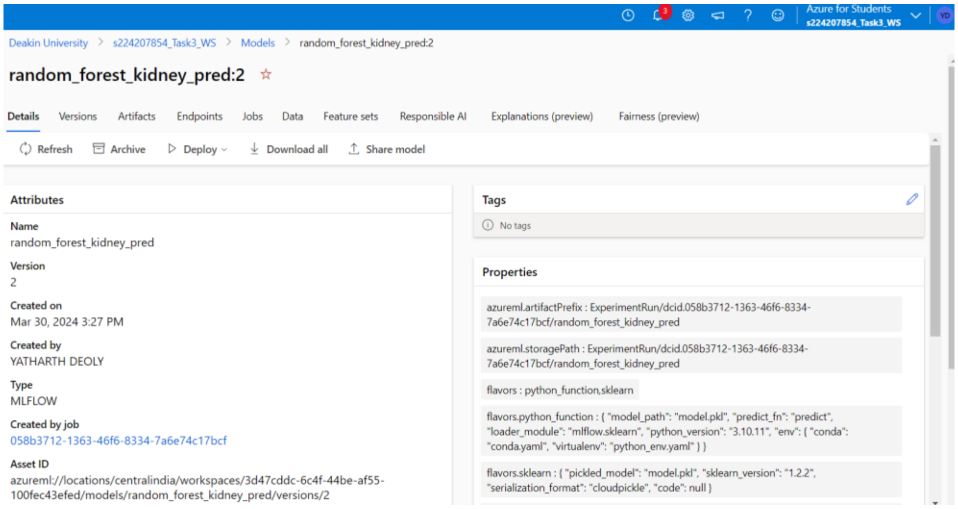

```python
mlflow.start_run()
run = experiment.start_logging()

random_forest_classifier = RandomForestClassifier(
    criterion="gini", max_depth=7, n_estimators=20, random_state=42
)

run.log("num_samples", kidney_disease_df.shape[0])
run.log("num_features", kidney_disease_df.shape[1] - 1)
run.log("criterion", "gini")

# fit the model using fit() on train data
random_forest = random_forest_classifier.fit(X_train, y_train)
y_pred = random_forest.predict(X_test)

print("\nClassification Matrix:\n", classification_report(y_test, y_pred))

# Logging all metrics of classification_report
cr = classification_report(y_test, y_pred, output_dict=True)

run.log("accuracy", cr.pop("accuracy"))

for class_or_avg, metrics_dict in cr.items():
    for metric, value in metrics_dict.items():
        run.log(class_or_avg + "_" + metric, value)

model_name = "random_forest_kidney_pred"

print("Registering the model via MLFlow")
mlflow.sklearn.log_model(
    sk_model=random_forest_classifier,
    registered_model_name=model_name,
    artifact_path=model_name,
)

# Saving the model to a file
mlflow.sklearn.save_model(
    sk_model=random_forest_classifier,
    path=os.path.join(model_name, "trained_model"),
)

run.complete()

mlflow.end_run()
```

Model output

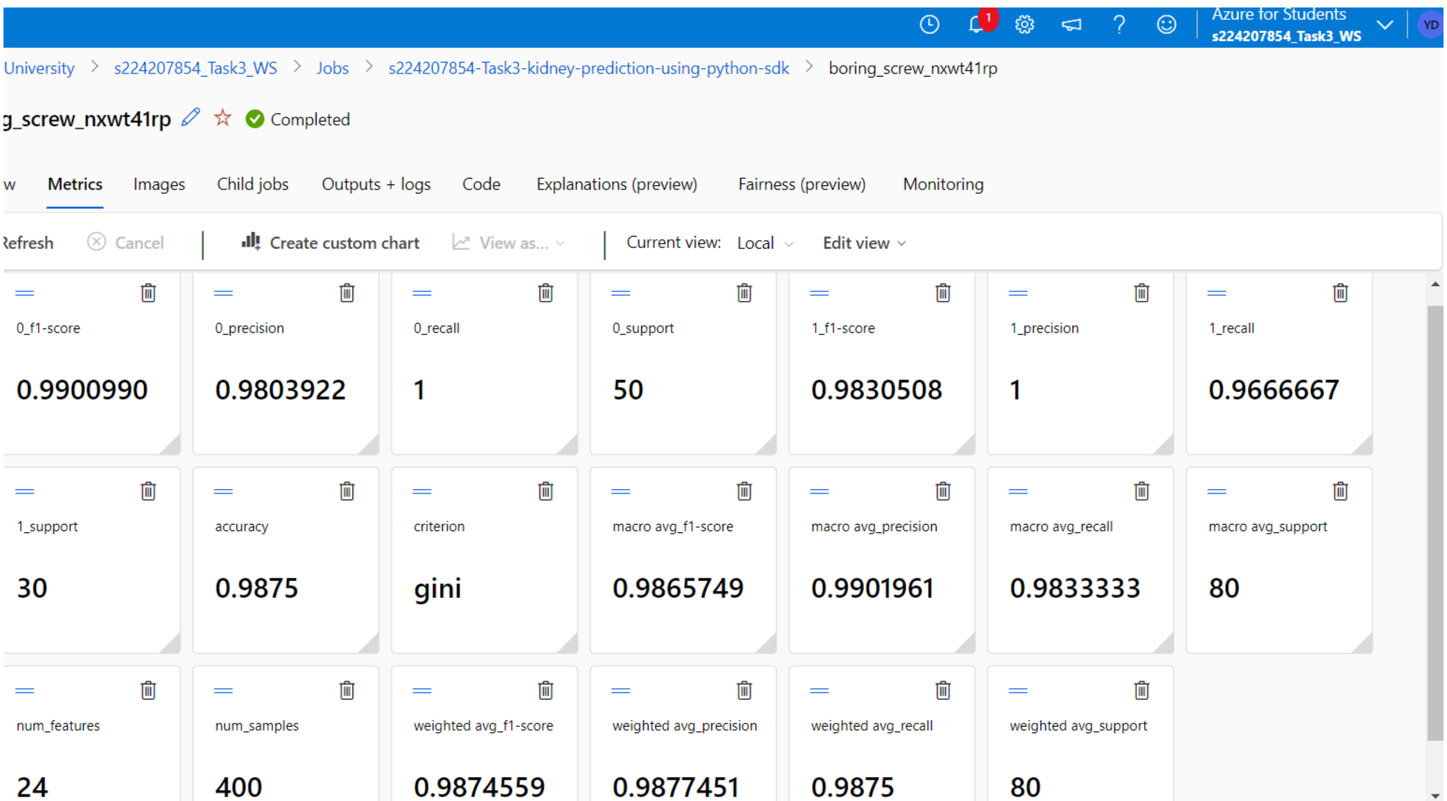



```
Registered model 'random_forest_kidney_pred' already exists. Creating a new version of this model...
2024/03/30 09:57:31 INFO mlflow.tracking._model_registry.client: Waiting up to 300 seconds for model
version to finish creation. Model name: random_forest_kidney_pred, version 2
Created version '2' of model 'random_forest_kidney_pred'.

Classification Matrix:
               precision    recall  f1-score   support

           0       0.98      1.00      0.99        50
           1       1.00      0.97      0.98        30

    accuracy                           0.99        80
   macro avg       0.99      0.98      0.99        80
weighted avg       0.99      0.99      0.99        80

Registering the model via MLFlow
```

Once model is registered, we can check same over Azure Portal in "Model List".

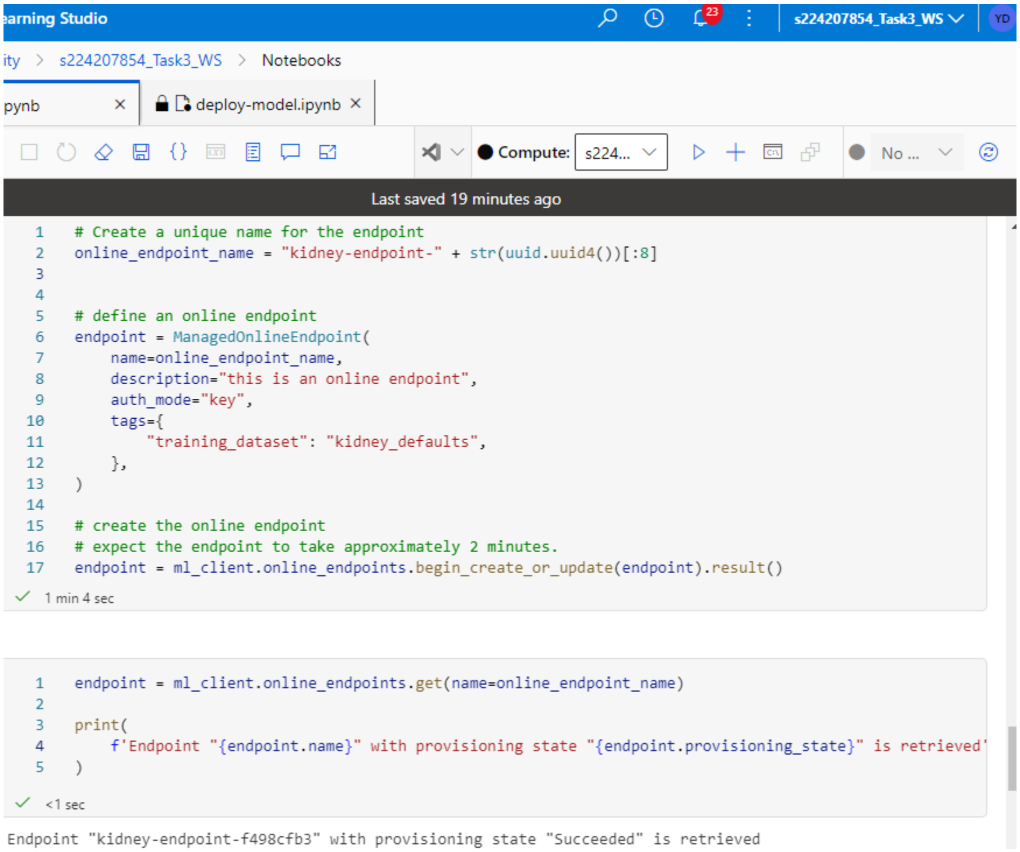


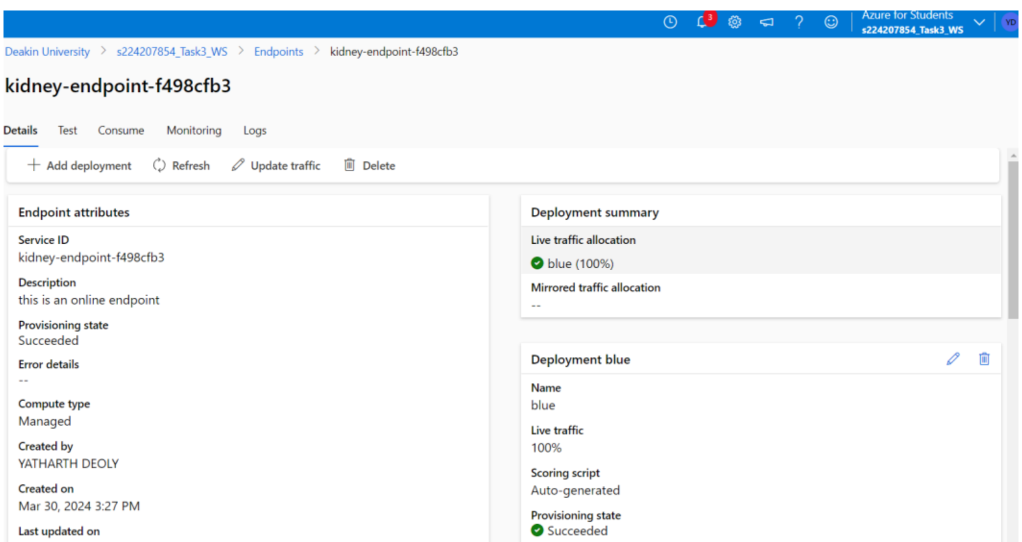
Model metrics can also be checked In Azure portal inside jobs section as we have logged all the metrics.



Endpoint Creation

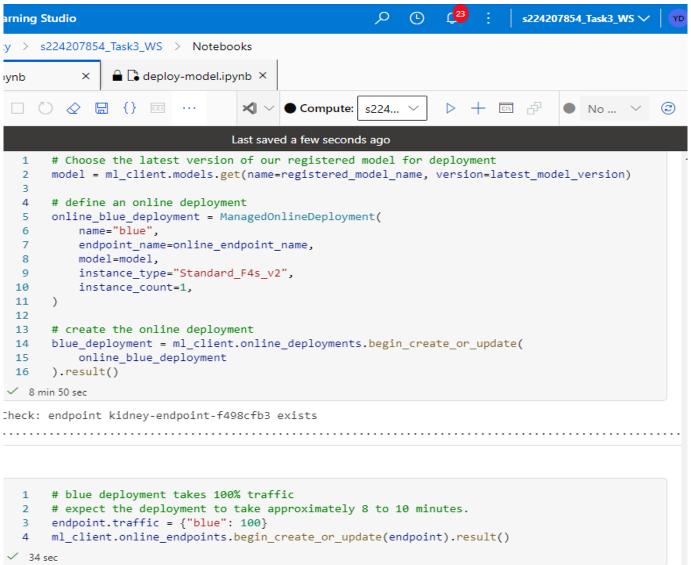
After model registration, we will be creating an endpoint which will be used for model deployment. Here we will be creating dynamic endpoint using 'uuid'. Once endpoint is up and running, we can check the status via coding way or checking over Azure ML portal.



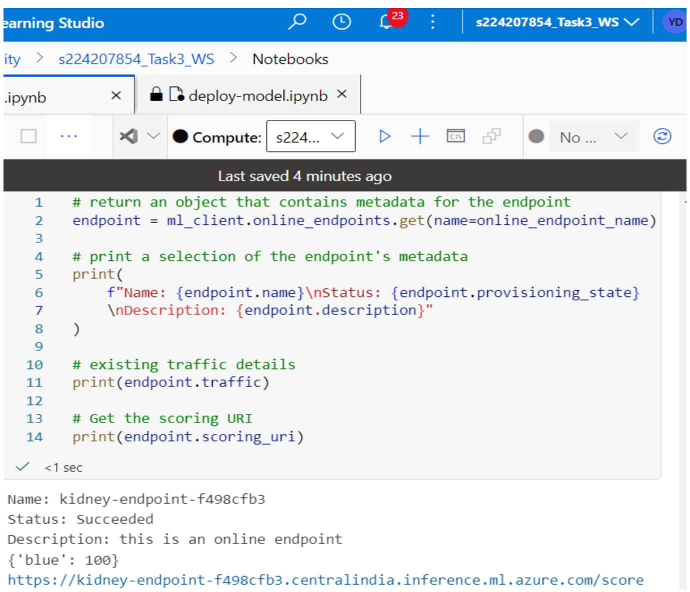


Model Deployment

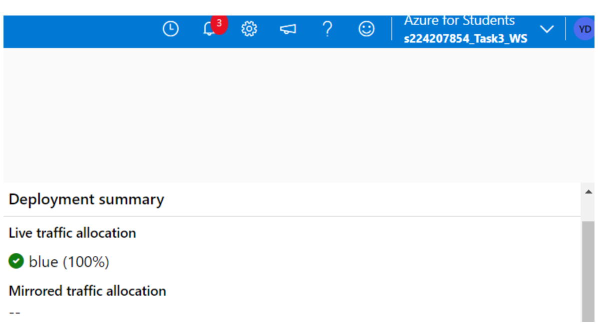
As our model and endpoint is ready, we will start our blue deployment with the latest version of model over endpoint. As we have created the cluster of instance type as “Standard_F4s_v2”, will be using the same instance for deployment also.



Checking the status of deployment over endpoint.



Deployment status can also be checked from Azure portal from “Endpoints”



Summary

In this task we have learnt about usage of Azure with machine learning and how to train, manage and deploy model in Azure ML workspace.

References

- https://olympus.mygreatlearning.com/courses/103482/modules/items/5568439
- https://learn.microsoft.com/en-us/azure/machine-learning/tutorial-train-model?view=azureml-api-2
- https://learn.microsoft.com/en-us/azure/machine-learning/tutorial-deploy-model?view=azureml-api-2