SIG788

Engineering AI solutions

Pass Task 3

Arunkumar Balaraman
S223919051

Contents

Target = Pass

Design and Deploy Model using Azure Machine Learning

Introduction

Chronic Kidney Disease (CKD) poses a significant and escalating global health issue. The early identification and prompt intervention can substantially modify the course of the disease, enhancing patient prognosis and minimizing healthcare expenses. The emergence of machine learning technologies has made it increasingly possible and precise to forecast the onset and progression of CKD based on diverse health indicators. This project seeks to leverage the capabilities of Azure Machine Learning (Azure ML) and the Python Software Development Kit (SDK) to create a predictive model specifically for this purpose.

Objectives:

The main objective of this project is to design, build, and implement a machine learning model that can accurately predict the existence or risk of CKD in individuals based on a variety of health parameters. This involves:

- **Data Preparation and Analysis**: Using the CKD dataset to investigate, cleanse, and prepare data for modelling.
- **Model Development**: Utilizing the capabilities of Azure ML to train and validate a predictive model.
- **Model Deployment**: Ensuring the model is available for real-time predictions through Azure ML's deployment services.

Azure Machine Learning SDK:

The **Azure Machine Learning SDK** for Python is a fundamental toolkit for executing machine learning tasks on the Azure platform. It offers a versatile and comprehensive structure that allows data scientists and machine learning engineers to automate and scale their machine learning processes using Azure's resources. The SDK aims to ease and expedite different phases of the machine learning lifecycle, encompassing data preparation, model training, model deployment, and monitoring.

Workspace:

Workspace in Azure Machine Learning SDK for Python is a fundamental resource for managing training and deployment artifacts. It serves as the central hub for machine learning operations, where we can experiment, train, and deploy machine learning models.

Each Workspace is tied to an Azure subscription and resource group, and it provides a secure and scalable cloud environment. It simplifies and streamlines various stages of the machine learning lifecycle, including data preparation, model training, model deployment, and monitoring.

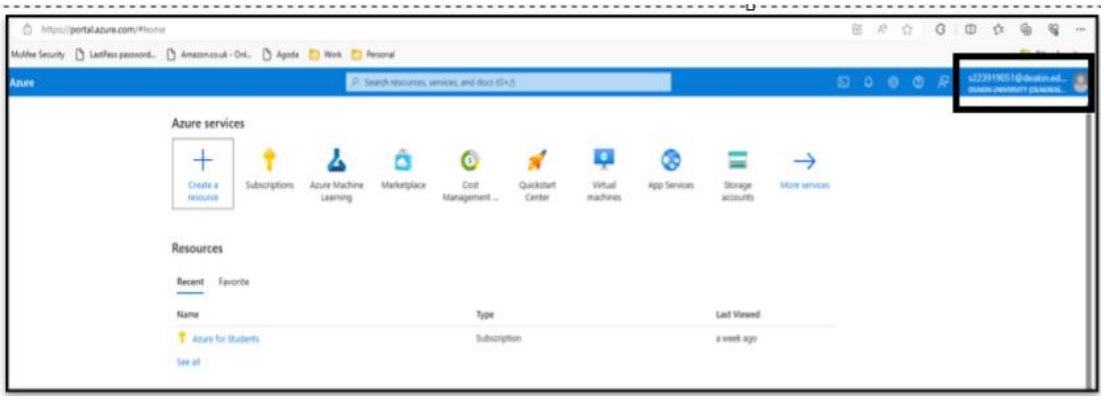
Workspace class in Azure ML SDK is a foundational resource in the cloud that use to manage all resources related to our machine learning workflow.

Creating Azure Machine Learning Workspace:

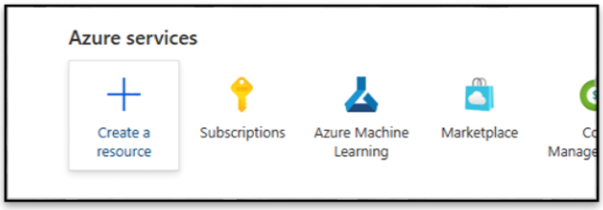
- **Sign in to Azure Portal:** Azure Portal and sign in with Microsoft account Deakin credentials.

  Link: https://portal.azure.com/

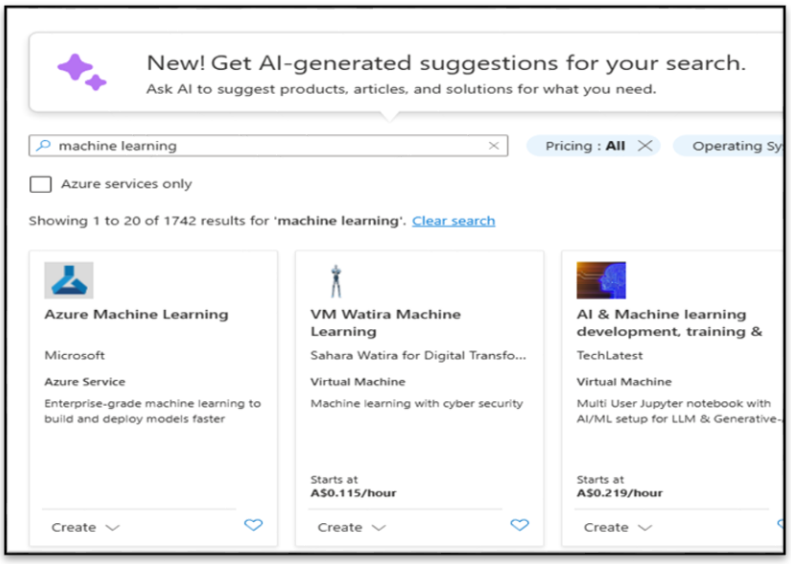
  Landing Page:



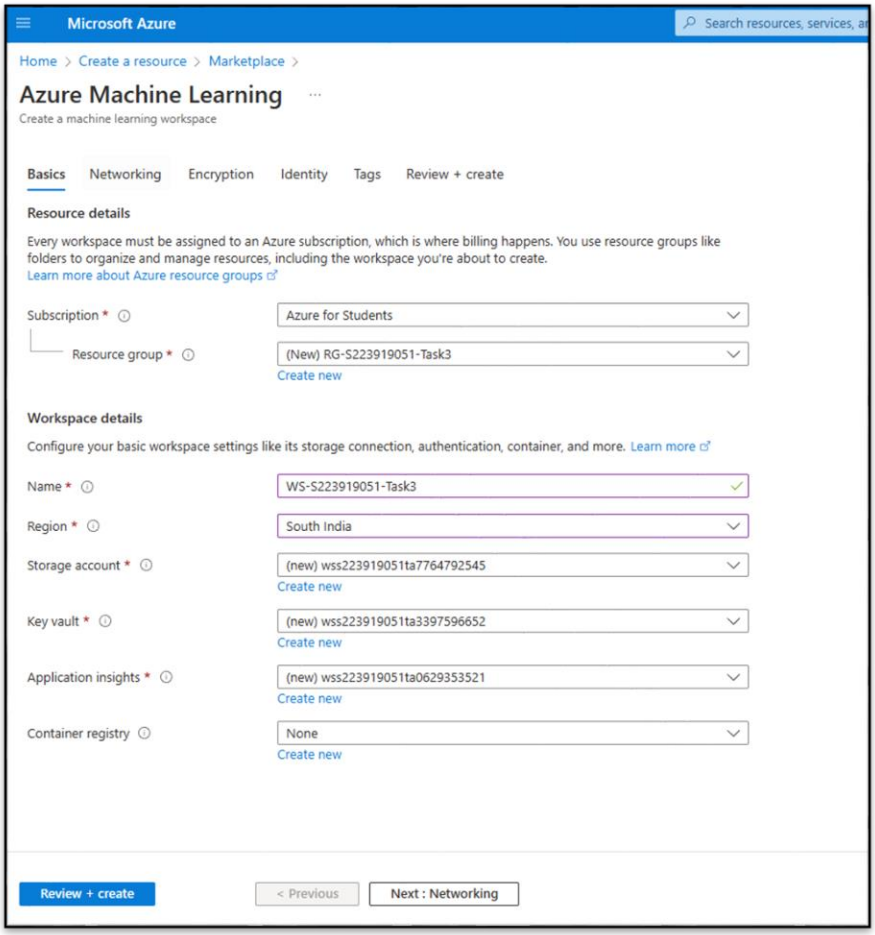
- **Create a New Workspace**:
  - Click on **Create a resource** at the top-left corner.



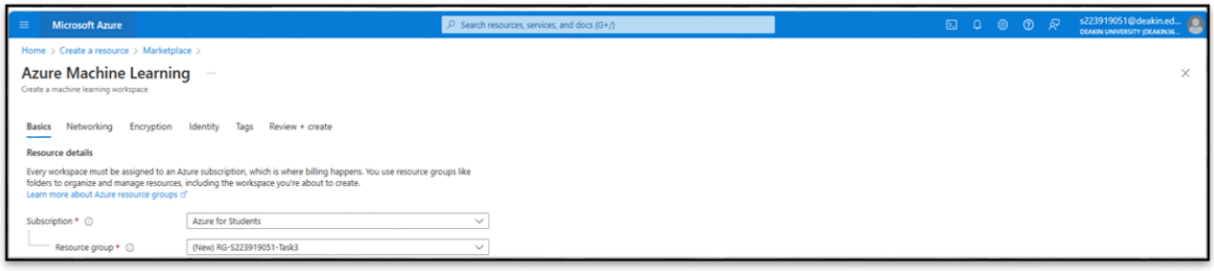
  - Search for **Machine Learning** & select **Create** from Azure Machine Learning to initiate the workspace creation process.



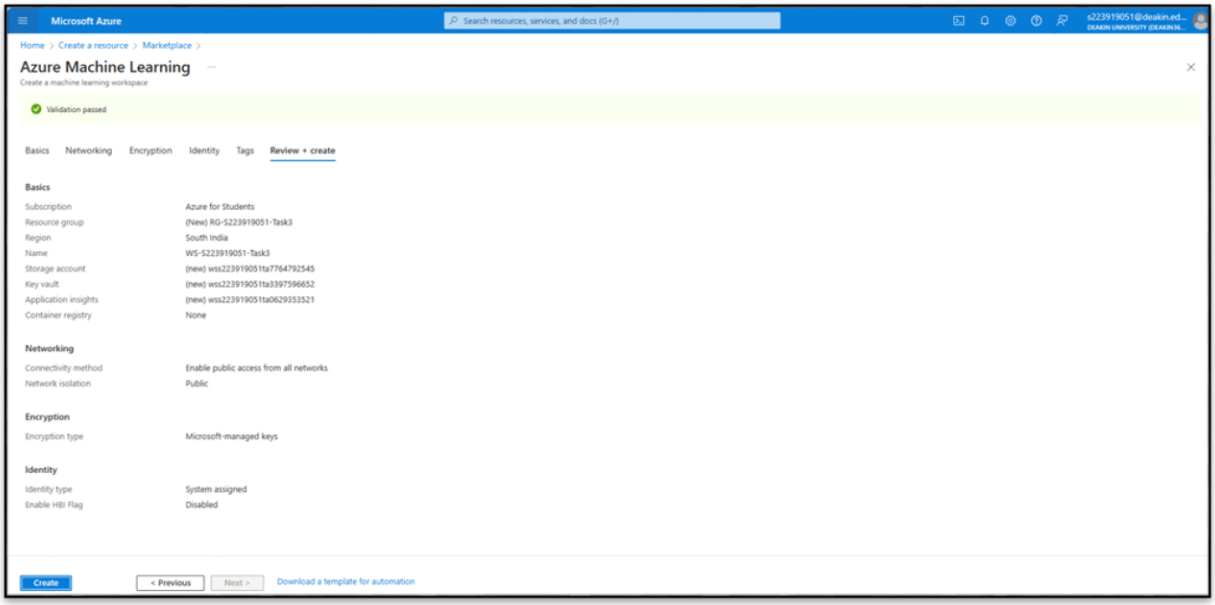
- **Configure the Workspace**:
  - Selecting Deakin Student Azure subscription
  - Creating a resource group **RG-S223919051-Task3**
  - Unique workspace name **WS-S223919051-Task3**
  - Choose an Azure region **South India** which is close to my location.
  - This process has automatically created a new
    - Azure Storage account,
    - Key Vault,
    - Application Insights.
    - Keeping the container registry as None for this project.



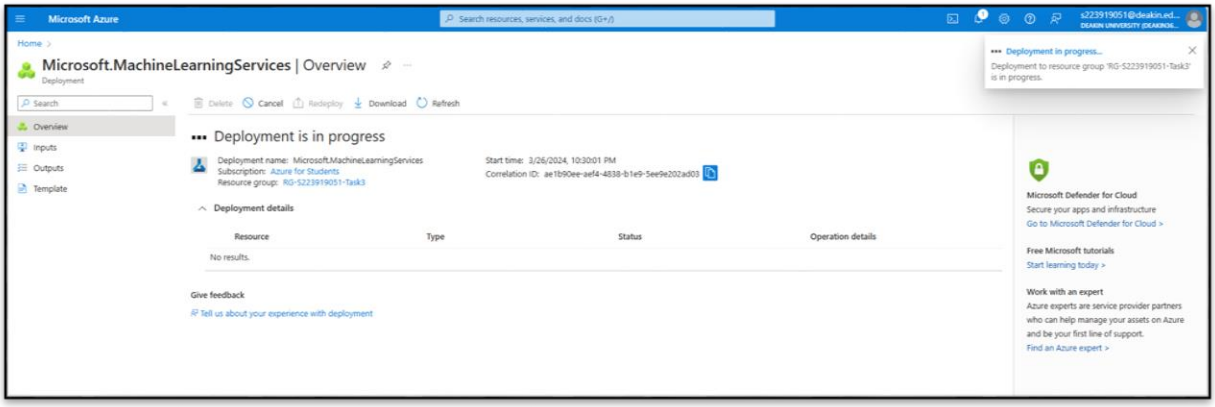
With My Student login ID:



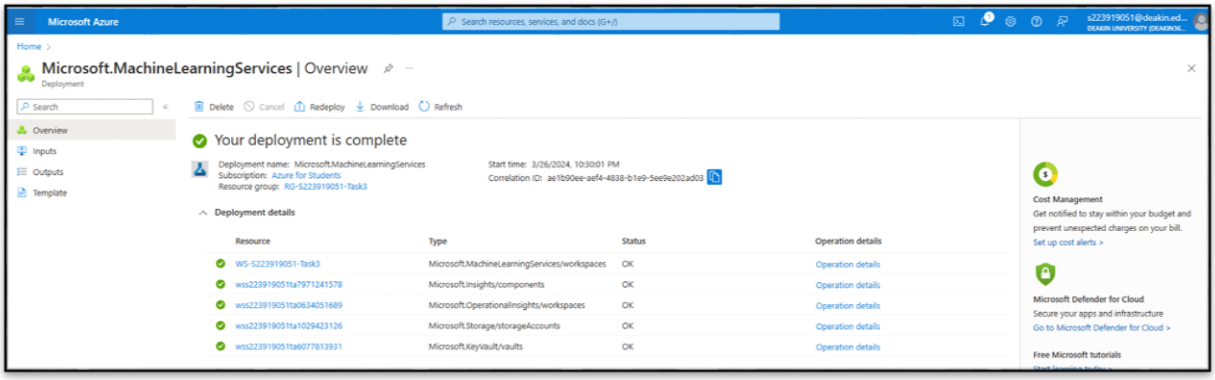
- **Review and Create**: After configuring the details click on **Review + create** button. Once Azure validates configuration. Click on **Create** button to deploy workspace.
  Review:



Creating:

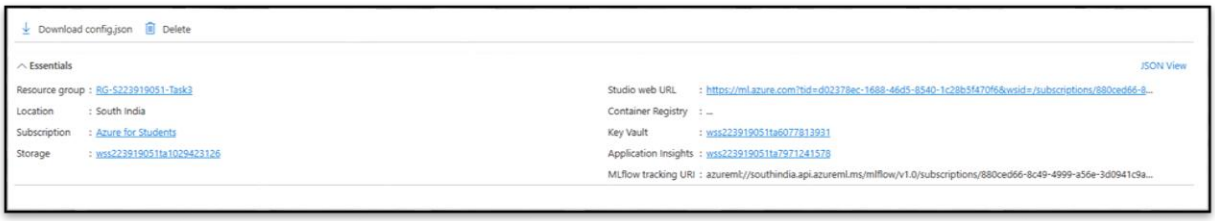


Workspace deployed:



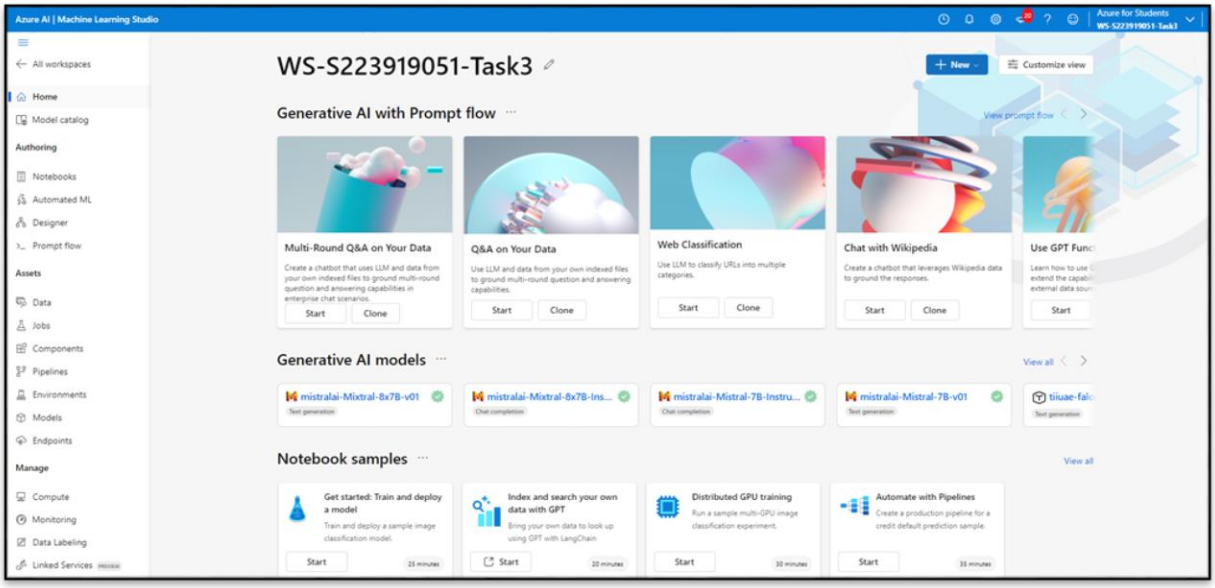
Final Workspace:

Link: https://ml.azure.com?tid=d02378ec-1688-46d5-8540-1c28b5f470f6&wsid=/subscriptions/880ced66-8c49-4999-a56e-3d0941c9ab71/resourcegroups/RG-S223919051-Task3/providers/Microsoft.MachineLearningServices/workspaces/WS-S223919051-Task3



- Access the Workspace: Once created, access workspace from the “All resources” tab or directly from the dashboard if pinned it. Inside, we’ll find various tools and options to start building and deploying machine learning models.

Accessing the Workspace:



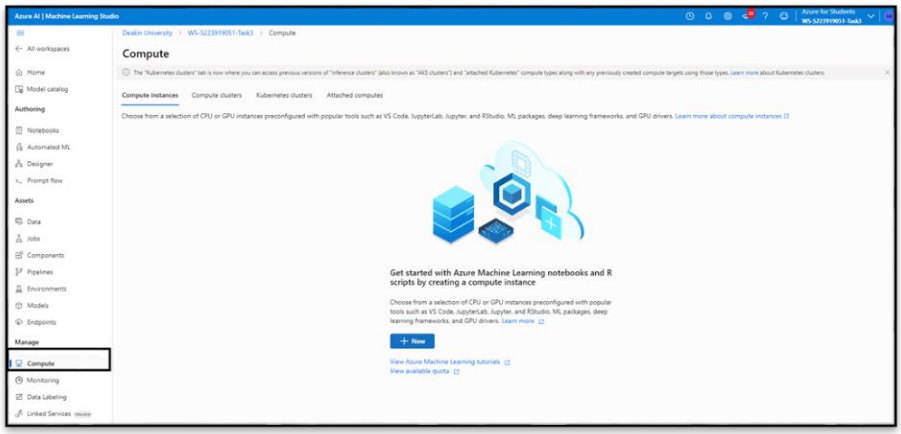
Now let’s create the compute instance and cluster:

Create Compute:

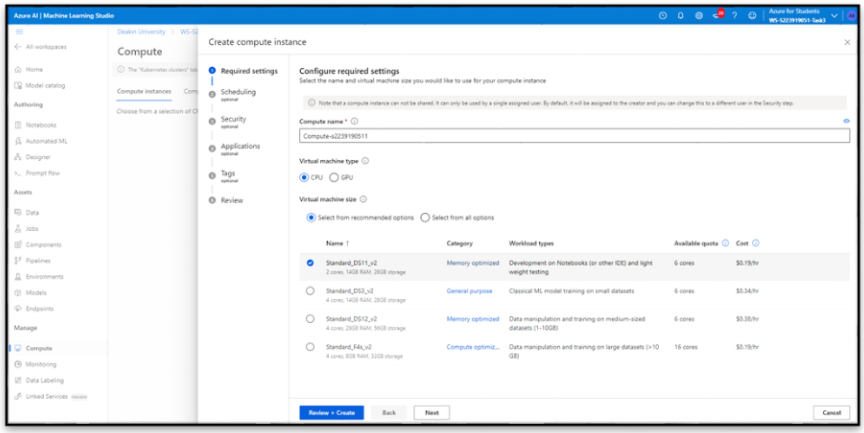
In Azure Machine Learning Studio:

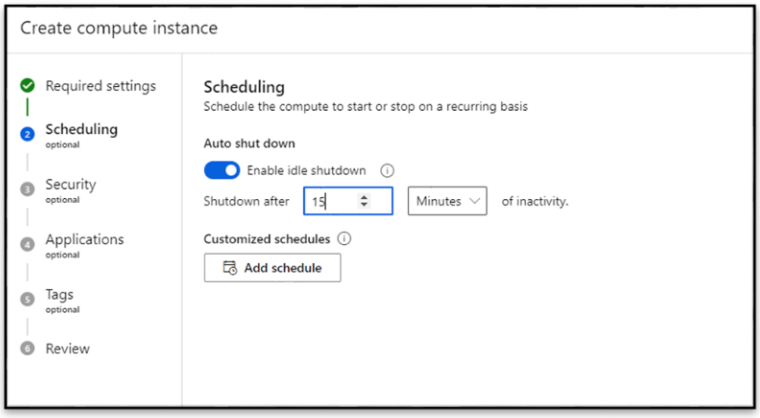
- A **compute target** is a specified compute resource or environment for running training scripts or hosting service deployments. It can be a local machine or a cloud-based resource.
- **Compute** targets allow easy switching of compute environments without code modification and can be reused across training jobs.
- Azure ML supports various **compute** targets. Typically, we start with a local environment for small data experiments, then **scale up** or use distributed training with training compute targets. After the model is ready, it's deployed to a web hosting environment with deployment compute targets.
- The **compute resources** used for compute targets are attached to a workspace and, except for the local machine, are shared by workspace users.

Open Compute: In the ML studio, click on **Compute** in the left-hand menu.
Create Compute Instance: In the compute instances tab, click **+ New**.

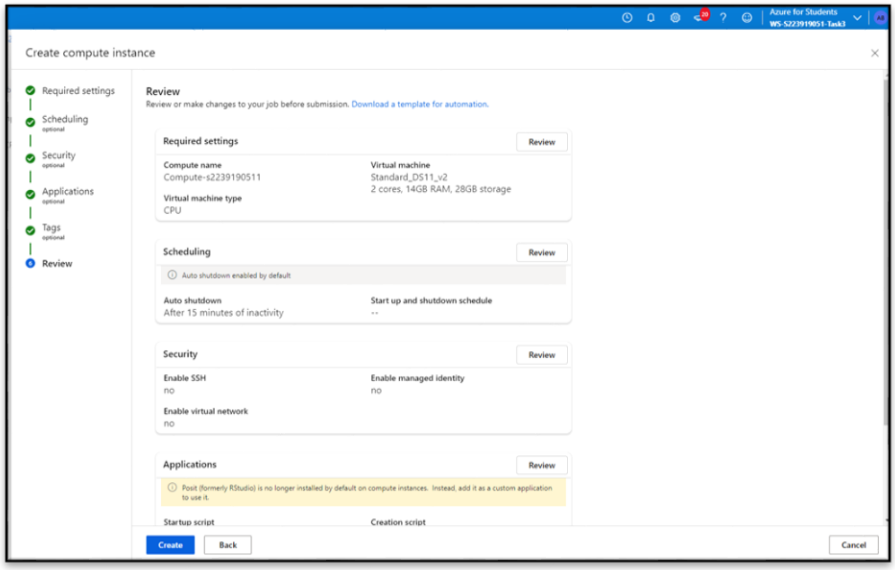


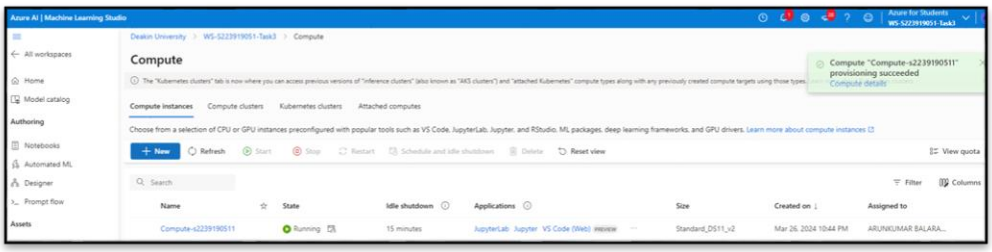
Configure: Enter a name for instance **Compute-s2239190511** & selected the a virtual machine size with cost $0.19 USD & click Next to schedule to shut down after 15 minutes





Create: **Review** and then Click on **Create** to start the provisioning of compute instance.





Create a Compute Cluster:

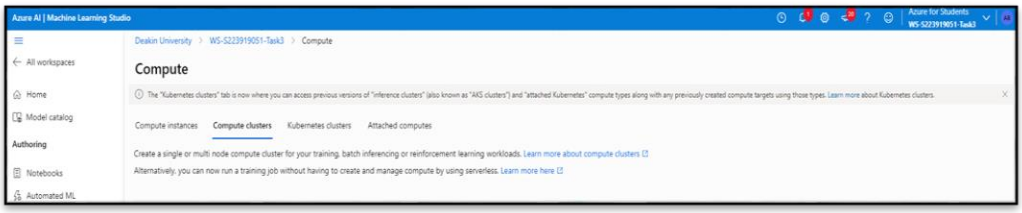
It's a **managed-compute infrastructure** in Azure Machine Learning Studio that allows creation of single or multi-node compute. It scales up automatically when a job is submitted and can be put in an Azure Virtual Network

Shared Resource: The compute cluster can be shared with other users in workspace.

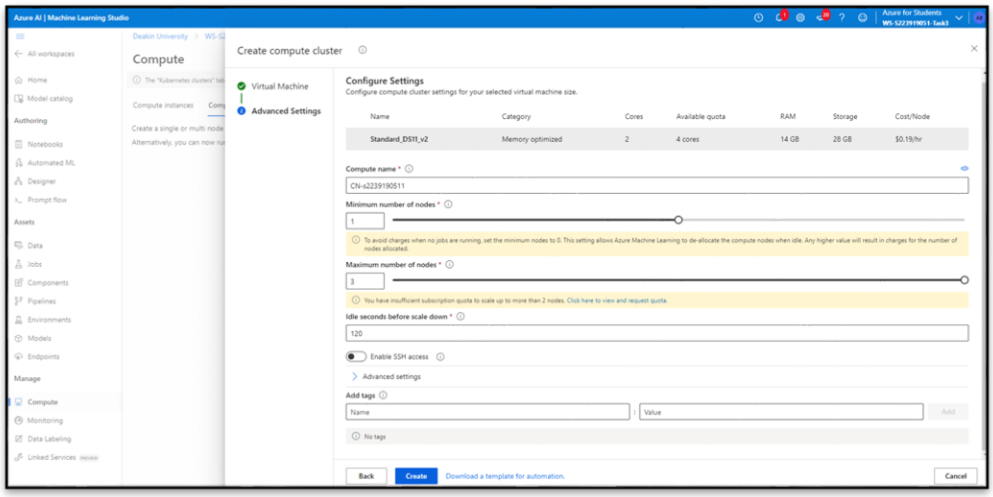
Adjusting Nodes: **Compute clusters** auto-scale between the minimum and maximum node count depending on the number of jobs submitted. By setting minimum nodes to 0, the cluster scales to 0 when there are no running jobs, saving costs

Now let's create the compute

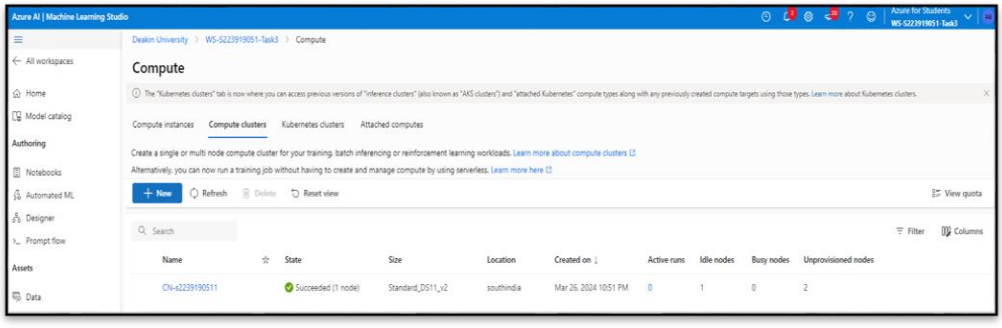
Navigate to Compute Clusters: In the **Compute** section, switch to the **Compute clusters** tab.



New Compute Cluster: Click **+ New** to create a new compute cluster.
Configure: Provide a name, choose virtual machine size, specify the minimum and maximum number of nodes, and configure other settings according to requirements.
Create: Click "Create" to provision compute cluster.



Created:



As now we have the set-up our workspace, let's now proceed with the datasets and review how to use Azure Machine learning **Python SDK** to train and deploy using Python Code including creating Resource Group, Workspace, Compute and Compute cluster.

Pre-requisite Libraries for Azure Machine learning Python SDK:

Installed below libraries for Azure ML services using pip install in Anaconda prompt.

- **azure-ai-ml**: A library for managing the lifecycle of machine learning models in the Azure cloud, from experimentation to production.
- **azure-identity**: A set of Python classes for secure credential management when connecting to Azure resources.
- **azureml-core**: A key component of the Azure Machine Learning service, this library automates model training, deployment, monitoring, and management.
- **mlflow**: An open-source platform for end-to-end machine learning lifecycle management, supporting multiple ML libraries and tools.

As we already demonstrated how to create the workspace, compute, compute cluster using Azure portal. we are using those resources to train & deploy the model.

Step1: Import all necessary Libraries for using Python SDK for CKD classification

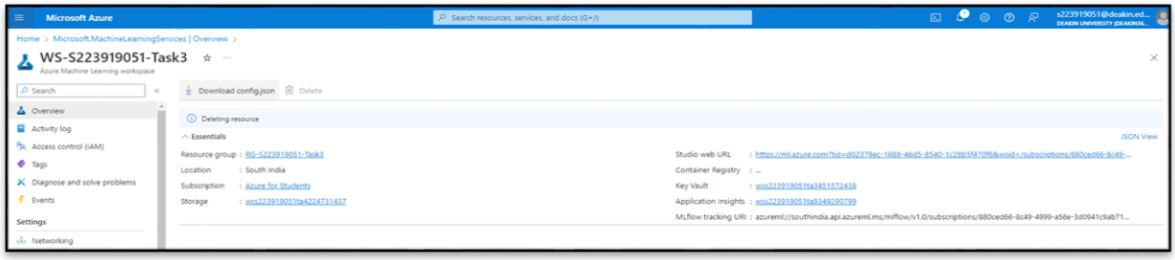
```
# Standard library imports
import os
import warnings  # To suppress warnings

# Suppress all warnings
warnings.filterwarnings('ignore')  # Suppress all warnings

# Data manipulation and numerical computing
import pandas as pd
import numpy as np

# Machine Learning - Model Training and Evaluation
from sklearn.model_selection import train_test_split  # Split arrays or matrices into random train and test subsets
from sklearn.ensemble import RandomForestClassifier  # Classifier for random forest models
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix  # Classification metrics
from sklearn.pipeline import Pipeline  # Pipeline to assemble steps for data transformation and model fitting
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder  # Data preprocessing
from sklearn.compose import ColumnTransformer  # Applies transformers to columns of an array or pandas DataFrame
from sklearn.model_selection import RandomizedSearchCV  # Randomized search on hyper parameters
from sklearn.feature_selection import mutual_info_classif  # Mutual information for feature selection

# Data Visualization
import matplotlib.pyplot as plt  # Plotting library
import seaborn as sns  # Statistical data visualization
```

```
# Statistical Analysis
from scipy.stats import hmean, gmean  # Harmonic and geometric mean calculations

# Azure Machine Learning
from azureml.core import Workspace, Experiment  # Azure ML classes for managing workspace and experiments
from azureml.core.authentication import InteractiveLoginAuthentication

# MLflow for experiment tracking
import mlflow  # Open source platform for the machine learning lifecycle
import mlflow.sklearn  # MLflow integration for Scikit-Learn
from azureml.core import Workspace
import mlflow

# Custom Transformer for Data Preprocessing
from sklearn.base import BaseEstimator, TransformerMixin  # Base class for custom transformer
```

1. **azureml.core**: Manages Azure Machine Learning workspace and experiments.
2. **mlflow**: Manages the machine learning lifecycle.
3. **mlflow.sklearn**: Provides integration for Scikit-Learn.
4. **RandomForestClassifier**: Creates random forest models.
5. **sklearn.metrics**: Provides the classification_report for model evaluation.
6. **pandas**: Used for data manipulation and analysis.
7. **train_test_split**: Splits data into training and testing sets.

Step2: Login to Azure Portal to create workspace and download config file

As we already provided steps and screenshot on how to produce the workspace and resource, we are directly providing steps to download the config file and use the workspace in python SDK.



config.json file content:

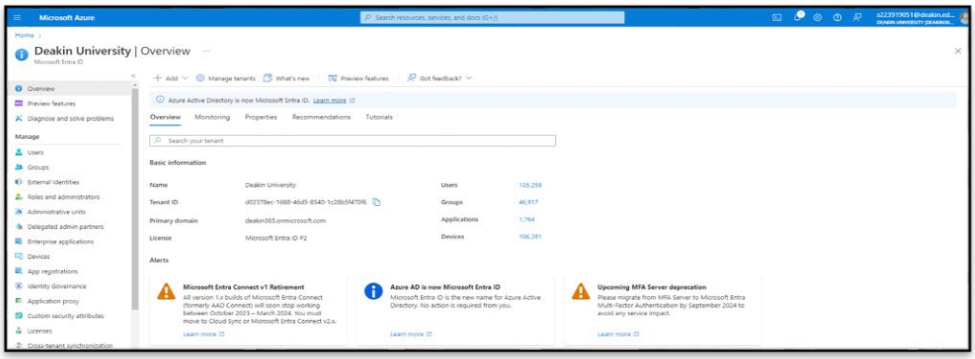
```
{
   "subscription_id": "880ced66-8c49-4999-a56e-3d0941c9ab71",
   "resource_group": "RG-S223919051-Task3",
   "workspace_name": "WS-S223919051-Task3"
}
```

Then place the config file in working directory.

Reading the config file:

```
In [4]: # Read the Config file
        ws = Workspace.from_config("config.json")
```

The configuration is accessed using interactive authentication via the Python SDK. This process also necessitates obtaining the tenant ID. The tenant ID can be retrieved from the Azure portal using the Entra ID.



Retrieve the tenant ID from the Entra and insert it into the code for interactive authentication using the Python SDK.

```
# Interative Authentication
interactive_auth = InteractiveLoginAuthentication(tenant_id="d02378ec-1688-46d5-8540-1c28b5f470f6")
```

Successfully authenticated and below is the header of the authentication from python sdk.

```
In [7]: # get_authentication_header
        interactive_auth.get_authentication_header()

Out[7]: {'Authorization': 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6InEtMjNmYWxldlpoaEQzaG05Q1Fia1A1TVF5VSIsImtpZCI6InEtMjNmY
Wxldlpoa EQzaG05Q1Fia1A1TVF5VSJ9.eyJhdWQiOiJodHRwczovL21hbmFnZW1lbnQuY29yZS53aW5kb3dzLm5ldC8iLCJpc3MiOiJodHRwczovL3N0cy53aW5kb3d
zLm5ldC9kMDIzNzhlYy0xNjg4LTQ2ZDUtODU0MC0xYzI4YjVmNDcwZjYvIiwiaWF0IjoxNzExNzMzNTcwLCJuYmYiOjE3MTE3MzM1NzAsImV4cCI6MTcxMTczODQxMC
wiYWNyIjoiMSIsImFpbyI6IkFWUUFxLzhXQUFBQTdZZytkU3YwbCtoRWxxY093cG5SS1dLVkJnNkNjUWd0T05NZVpDdjZySTlVclpzT0Z3a3pzMlF2UVprSk9KVWJ3S
0pob3ZoZUF4cytPY1pGQUhVczRlTzBKRDdWTFZtZjFsbUJzY2g4QkJFPSIsImFtciI6WyJwd2QiLCJtZmEiXSwiYXBwaWQiOiIwNGIwNzc5NS04ZGRiLTQ2MWEtYmJl
ZS0wMmY5ZTFiZjdiNDYiLCJhcHBpZGFjciI6IjAiLCJmYW1pbHlfbmFtZSI6IkJBTEFSQU1BTiIsImdpdmVuX25hbWUiOiJBUlVOS1VNQVIiLCJncm91cHMiOlsiZmI
5NmU4MTYtNzFlOS00OGE1LTljY2ItODkwZDc4MThhZDBlIiwiZTk3NTQ3MWYtYWZkNC00NzE4LWJkMTktYjJiMmMwMGM5ZjU3IiwiYTNlM2U0NDItMDNjNC00ODk1LT
kyOWItNmJiZDIwYTVkMTIyIiwiNmYxZTlmODItYmJmMC00OWI3LTkyYzYtNmY2YmQzZjJhMWI4IiwiOGNjYTc0YjctYTJlMS00ODBjLWE4MmUtZWJlZDBkNDVkY2NmI
iwiNjY5NWZhYzctOGZlNC00OWMyLThkY2MtNDQ5Mjc4NDM3YjMwIiwiNTU1ZGQxZDQtZDE3My00YzM3LTg4OWQtM2JiNzgwMDIyZTc1IiwiYjc5NTgwZWEtNDY0Ny00
OTQ4LWI2NTEtMGM0NDhhNDk0ZjA4Il0sImlkdHlwIjoidXNlciIsImlwYWRkciI6IjI0MDE6NDkwMDoxY2UxOmQyOWE6YTFmNDo3ZDAxOjRiYmY6OTRiNCIsIm5hbWU
iOiJBUlVOS1VNQVIgQkFMQVJBTUFOIiwib2lkIjoiNDA5NTM0YzItNjZmZC00NDU4LThmZTgtNThkZTU2YjIzZDFjIiwib25wcmVtX3NpZCI6IlMtMS01LTIxLTI0OD
k2MzA1Ny02MTQxMDM2NjEtMzA2NzIzMjc5OS05NTYzODIiLCJwdWlkIjoiMTAwMzIwMDJCOEZERDM5NyIsInJoIjoiMC5BV2NBN0hnajBJZ1cxVWFGUUJ3b3RmUnc5a
1pJZjNrQXV0ZFB1a1Bhd2ZqMk1CTm5BS2cuIiwic2NwIjoidXNlcl9pbXBlcnNvbmF0aW9uIiwic3ViIjoiS1VDcTJERC1kSUJJQXNSc1pLMW9kcmFnWkdGTjlKREdt
OUdtby1RbzViOCIsInRpZCI6ImQwMjM3OGVjLTE2ODgtNDZkNS04NTQwLTFjMjhiNWY0NzBmNiIsInVuaXF1ZV9uYW1lIjoiczIyMzkxOTA1MUBkZWFraW4uZWR1LmF
1IiwidXBuIjoiczIyMzkxOTA1MUBkZWFraW4uZWR1LmF1IiwidXRpIjoiTHRBdEZPdHA2a0NaaExuSHpRUVFBQSIsInZlciI6IjEuMCIsIndpZHMiOlsiYjc5ZmJmNG
QtM2VmOS00Njg5LTgxNDMtNzZiMTk0ZTg1NTA5Il0sInhtc190Y2R0IjoxMzc4NzkxNzk5fQ.Pg-pbpkigXxolM1vuDP6OkgGhk4n4DgNWsat6oS0t3adblgeA0ZXVP
XqcLeNb4wbF3z8tC7GdLVd0BvC8hYHND2GXa6Dwzim4XpORHKhK4TO681FReUUTgZW96lX2UMVcw91cO9j-5VkPRvx_d3GAjtDXSJJHfDqwRzec5pRWXHCUUevXhDcW
Lme5bsGXG2byBpLdmHHngkBV9wAK7zgxgI3x8aHQJUCMncIuqS_4akzSvO9bCEgK3nhBZ7c6_rfs58ci3uLai9ruV8-J_Jyj37yyzhqcB8cp7iZiWtXHOu5xB3DOOwr
d2ZstX4uhs24xLbqtwmpA-MSh_E8P6vRDg'}
```

Dataset:

Now let's clean the dataset and prepare train test split

During the initial data loading, several issues were encountered and resolved by **removing tab characters**, correcting **spacing around "yes" and "no"** values, and **fixing duplicated commas**. The cleaned dataset, stored in a new file named modified_chronic_kidney_disease.arff.

Code:

```
In [3]: # Setting Up File to read
        file_path = 'chronic_kidney_disease.arff'

        # Read the file
        with open(file_path, 'r') as file:
            file_content = file.readlines()

        # Applying Correction to the data
        file_content = [line.replace('\t', '') for line in file_content]  # Replace all tabs with nothing
        file_content = [line.replace(' yes', 'yes').replace('yes ', 'yes') for line in file_content]  # Correct 'yes'
        file_content = [line.replace(' no', 'no').replace('no ', 'no') for line in file_content]  # Correct 'no'
        file_content = [line.replace(',,', ',') for line in file_content]  # Correct ",,"

        # Save the modified file
        modified_file = 'modified_chronic_kidney_disease.arff'
        with open(modified_file, 'w') as file:
            file.writelines(file_content)
```

Observation:

Following steps are performed to fix the same:

- Removing Tab Characters: All tab characters were eliminated from the file to ensure uniform delimiter usage across the dataset.
- Correcting “Yes” and “No” Values Spacing: Rectified spacing problems around “yes” and “no” values.
- Fixing Comma Duplication: Handled occurrences of repeated commas which misrepresented the structure of the dataset.

By making these adjustments, stored the refined data in a new file **modified_chronic_kidney_disease.arff**. The cleaned dataset has been loaded for further analysis & which has **400 rows** and **25 columns**.

Data Preparation

Code:

```
In [4]: # load the ARFF file
        data, meta = arff.loadarff(modified_file)
        # Convert the data to a pandas DataFrame
        df = pd.DataFrame(data)
        # Convert byte strings to regular strings for object type columns
        df_obj = df.select_dtypes([object])
        df[df_obj.columns] = df_obj.apply(lambda x: x.str.decode('utf-8'))
        # top 5 records
        df.head()
```

Out[4]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1	0	?	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4	0	?	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2	3	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2	0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

5 rows × 25 columns

Observation:

The dataset has **400 rows** and **25 columns** and with size of **10,000**. As indicated in the instructions document "**chronic_kidney_disease.info.txt**".

The DataFrame has **null values** & few null values are represented as **'?'** which has been replaced with **np.nan**. The missing values are **imputed** with ***median*** *for numerical data* and ***mode*** *for categorical data*.

Below is the brief overview of the columns and their respective null percentages:

Age: 2.25%
Blood Pressure (bp): 3.00%
Specific Gravity (sg): 11.75%
Albumin (al): 11.50%
Sugar (su): 12.25%
Red Blood Cells (rbc): 38.00%
Pus Cell (pc): 16.25%
Pus Cell Clumps (pcc): 1.00%
Bacteria (ba): 1.00%
Blood Glucose Random (bgr): 11.00%
Blood Urea (bu): 4.75%
Serum Creatinine (sc): 4.25%
Sodium (sod): 21.75%
Potassium (pot): 22.00%
Hemoglobin (hemo): 13.00%
Packed Cell Volume (pcv): 17.75%
White Blood Cell Count (wbcc): 26.50%
Red Blood Cell Count (rbcc): 32.75%
Hypertension (htn): 0.50%
Diabetes Mellitus (dm): 0.50%
Coronary Artery Disease (cad): 0.50%
Appetite (appet): 0.25%
Pedal Edema (pe): 0.25%
Anemia (ane): 0.25%
Class: 0.00%

No duplicates were found in the DataFrame **before or after** the imputation.

Exploratory Data Analysis

Univariate Analysis

Describe

```
In [16]: df.describe().T
```

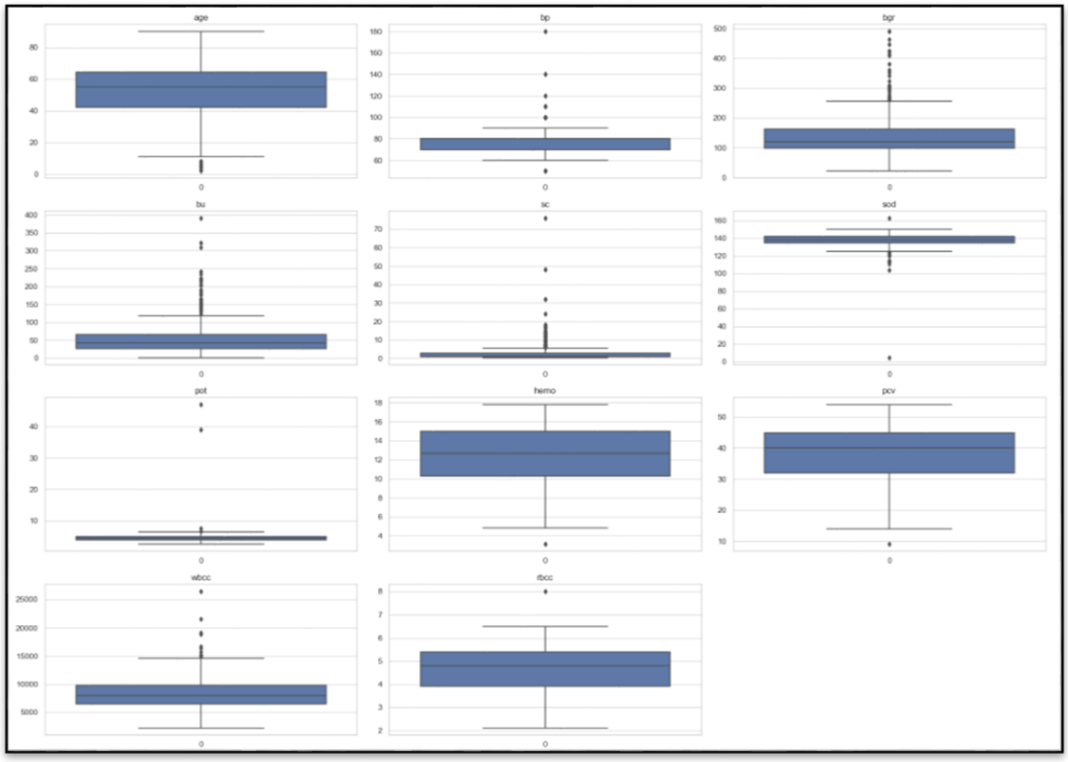
Out[16]:

	count	mean	std	min	25%	50%	75%	max
age	400.0	51.562500	16.982996	2.0	42.000	55.00	64.000	90.0
bp	400.0	76.575000	13.489785	50.0	70.000	80.00	80.000	180.0
bgr	400.0	145.062500	75.260774	22.0	101.000	121.00	150.000	490.0
bu	400.0	56.693000	49.395258	1.5	27.000	42.00	61.750	391.0
sc	400.0	2.997125	5.628886	0.4	0.900	1.30	2.725	76.0
sod	400.0	137.631250	9.206332	4.5	135.000	138.00	141.000	163.0
pot	400.0	4.577250	2.821357	2.5	4.000	4.40	4.800	47.0
hemo	400.0	12.542500	2.716490	3.1	10.875	12.65	14.625	17.8
pcv	400.0	39.082500	8.162245	9.0	34.000	40.00	44.000	54.0
wbcc	400.0	8298.500000	2529.593814	2200.0	6975.000	8000.00	9400.000	26400.0
rbcc	400.0	4.737750	0.841439	2.1	4.500	4.80	5.100	8.0

- The dataset represents a wide age range **(2-90 years),** with an average of **51.48** years.
- Blood pressure varies among individuals, with a mean of **76.47 mmHg**.
- Specific Gravity (sg) averages at **1.017**, indicating mostly normal kidney function. However, Albumin (al) and Sugar (su) levels vary widely **(0-5)**, suggesting varied kidney function.
- Blood Glucose Random (bgr) levels range from **22-490 mg/dl**, indicating diverse metabolic states.
- Blood Urea (bu) and Serum Creatinine (sc) show **significant variation**, highlighting potential kidney function issues.
- Sodium (sod) and Potassium (pot) levels are **within normal ranges**, but variability suggests potential **imbalances**.
- Hemoglobin (hemo), Packed Cell Volume (pcv), and White (wbcc) and Red Blood Cell Counts (rbcc) show wide ranges, important for diagnosing conditions like **anemia**.

Box Plot:

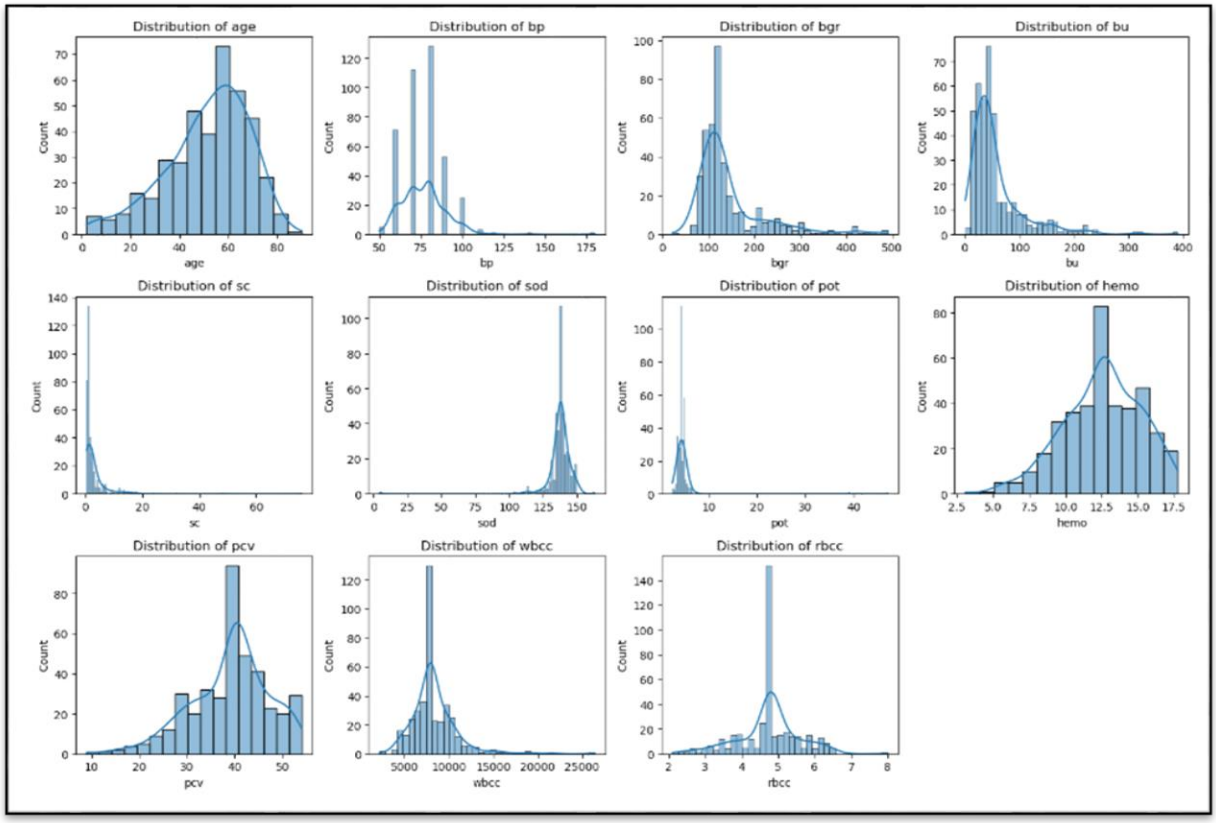
```
In [20]: # Setting the aesthetics for the plots
         sns.set(style="whitegrid")

         # Plotting box plots for numerical columns
         plt.figure(figsize=(20, 15))
         for i, col in enumerate(df_numerical, 1):
             plt.subplot(4, 3, i)
             sns.boxplot(data[col])
             plt.title(col)

         plt.tight_layout()
         plt.show()
```



Significant outliers in attributes such as bgr, bu, sc, sod, pot, and wbcc reflects natural variation or health conditions.
The IQR differences across parameters like bu and sc denote high variability among individuals.

Histogram and Skewness

```
In [17]: describe_df = df.describe()
         summary_list = []

         for column in describe_df.columns:
             mean = describe_df[column]['mean']
             std = describe_df[column]['std']
             min_val = describe_df[column]['min']
             max_val = describe_df[column]['max']

             median = describe_df[column]['50%']
             if mean > median:
                 skewness = "right skewed"
             elif mean < median:
                 skewness = "left skewed"
             else:
                 skewness = "normally distributed"

             summary = f"{column}: has a mean of {mean:.2f} & standard deviation of {std:.2f}. Values range from {min_val} to {max_val} &
             summary_list.append(summary)

         summary_list

Out[17]: ['age: has a mean of 51.56 & standard deviation of 16.98. Values range from 2.0 to 90.0 & left skewed.',
          'bp: has a mean of 76.58 & standard deviation of 13.49. Values range from 50.0 to 180.0 & left skewed.',
          'bgr: has a mean of 145.06 & standard deviation of 75.26. Values range from 22.0 to 490.0 & right skewed.',
          'bu: has a mean of 56.69 & standard deviation of 49.40. Values range from 1.5 to 391.0 & right skewed.',
          'sc: has a mean of 3.00 & standard deviation of 5.63. Values range from 0.4 to 76.0 & right skewed.',
          'sod: has a mean of 137.63 & standard deviation of 9.21. Values range from 4.5 to 163.0 & left skewed.',
          'pot: has a mean of 4.58 & standard deviation of 2.82. Values range from 2.5 to 47.0 & right skewed.',
          'hemo: has a mean of 12.54 & standard deviation of 2.72. Values range from 3.1 to 17.8 & left skewed.',
          'pcv: has a mean of 39.08 & standard deviation of 8.16. Values range from 9.0 to 54.0 & left skewed.',
          'wbcc: has a mean of 8298.50 & standard deviation of 2529.59. Values range from 2200.0 to 26400.0 & right skewed.',
          'rbcc: has a mean of 4.74 & standard deviation of 0.84. Values range from 2.1 to 8.0 & left skewed.']
```



Histogram and Skewness:

- 'age: has a mean of 51.56 & standard deviation of 16.98. Values range from 2.0 to 90.0 & left skewed.',
- 'bp: has a mean of 76.58 & standard deviation of 13.49. Values range from 50.0 to 180.0 & left skewed.',
- 'bgr: has a mean of 145.06 & standard deviation of 75.26. Values range from 22.0 to 490.0 & right skewed.',
- 'bu: has a mean of 56.69 & standard deviation of 49.40. Values range from 1.5 to 391.0 & right skewed.',
- 'sc: has a mean of 3.00 & standard deviation of 5.63. Values range from 0.4 to 76.0 & right skewed.',
- 'sod: has a mean of 137.63 & standard deviation of 9.21. Values range from 4.5 to 163.0 & left skewed.',
- 'pot: has a mean of 4.58 & standard deviation of 2.82. Values range from 2.5 to 47.0 & right skewed.',
- 'hemo: has a mean of 12.54 & standard deviation of 2.72. Values range from 3.1 to 17.8 & left skewed.',
- 'pcv: has a mean of 39.08 & standard deviation of 8.16. Values range from 9.0 to 54.0 & left skewed.',
- 'wbcc: has a mean of 8298.50 & standard deviation of 2529.59. Values range from 2200.0 to 26400.0 & right skewed.',

- 'rbcc: has a mean of 4.74 & standard deviation of 0.84. Values range from 2.1 to 8.0 & left skewed.'

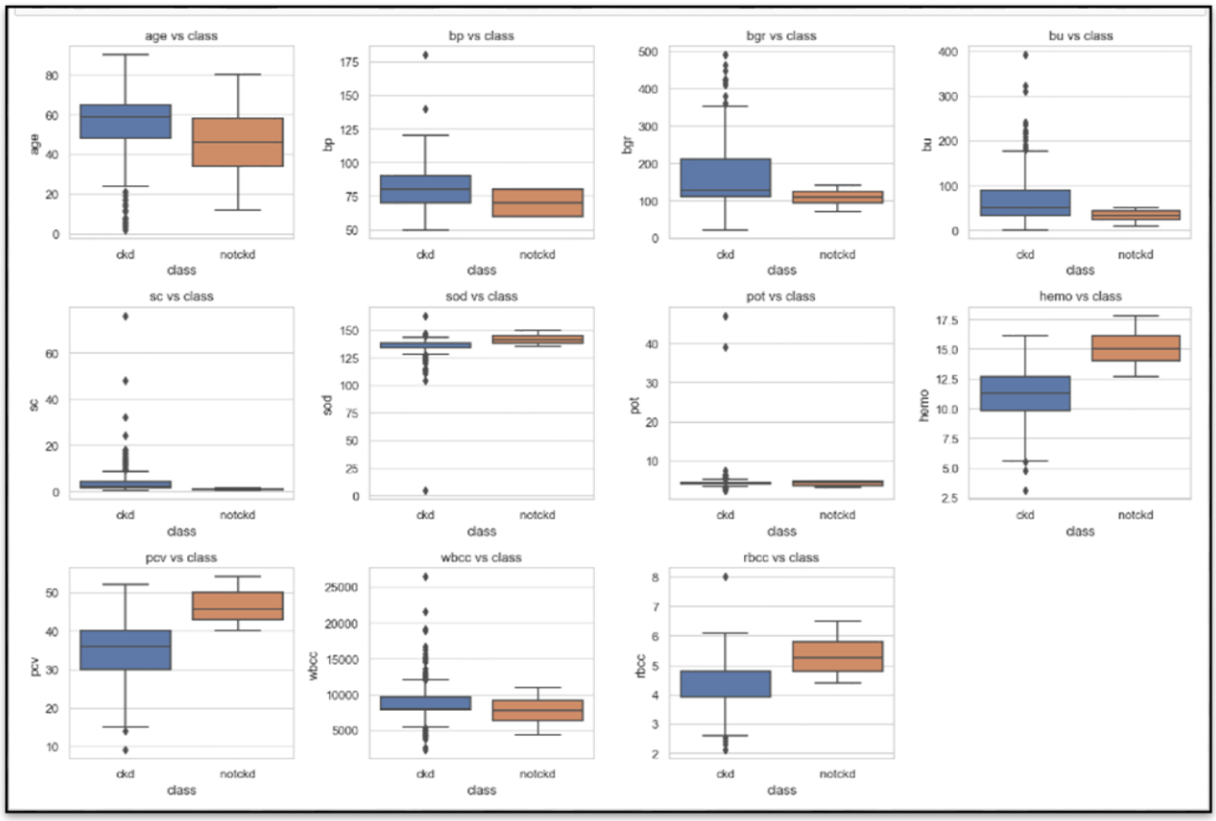
- Right-skewed distributions in al, su, bgr, bu, and sc indicate most values are low with a tail towards higher values. also, left-skewed distributions in age, bp, and hemo suggest a concentration of higher values.
- The non-normal distributions of sg, sod, and hemo reflect the complex of medical data.

Bivariate Analysis

Box Plot with Class

Code:

```
In [21]: # Plot Box plot
         plt.figure(figsize=(15,10))

         for i, column in enumerate(df_numerical.columns):
             plt.subplot(3, 4, i+1)
             sns.boxplot(x='class', y=column, data=df)
             plt.title(f'{column} vs class')

         plt.tight_layout()
         plt.show()
```

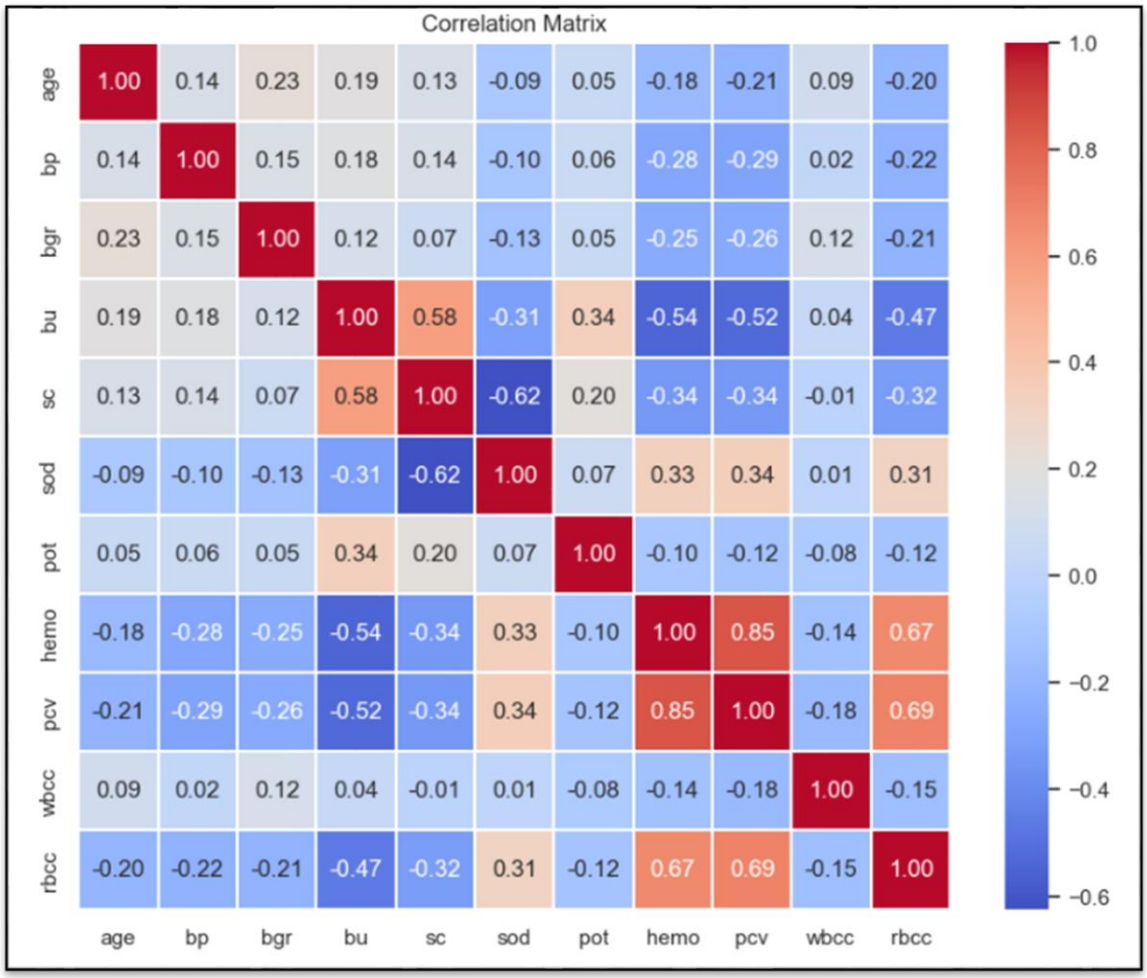


Box plots reveal distinct distributions for parameters like hemo, and pcv when compared against the disease presence class, suggesting these variables are key in distinguishing between classes.

Co-relation Analysis

Code:

```
In [22]: # Correlation Analysis
corr = df_numerical.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.05)
plt.title('Correlation Matrix')
plt.show()
```

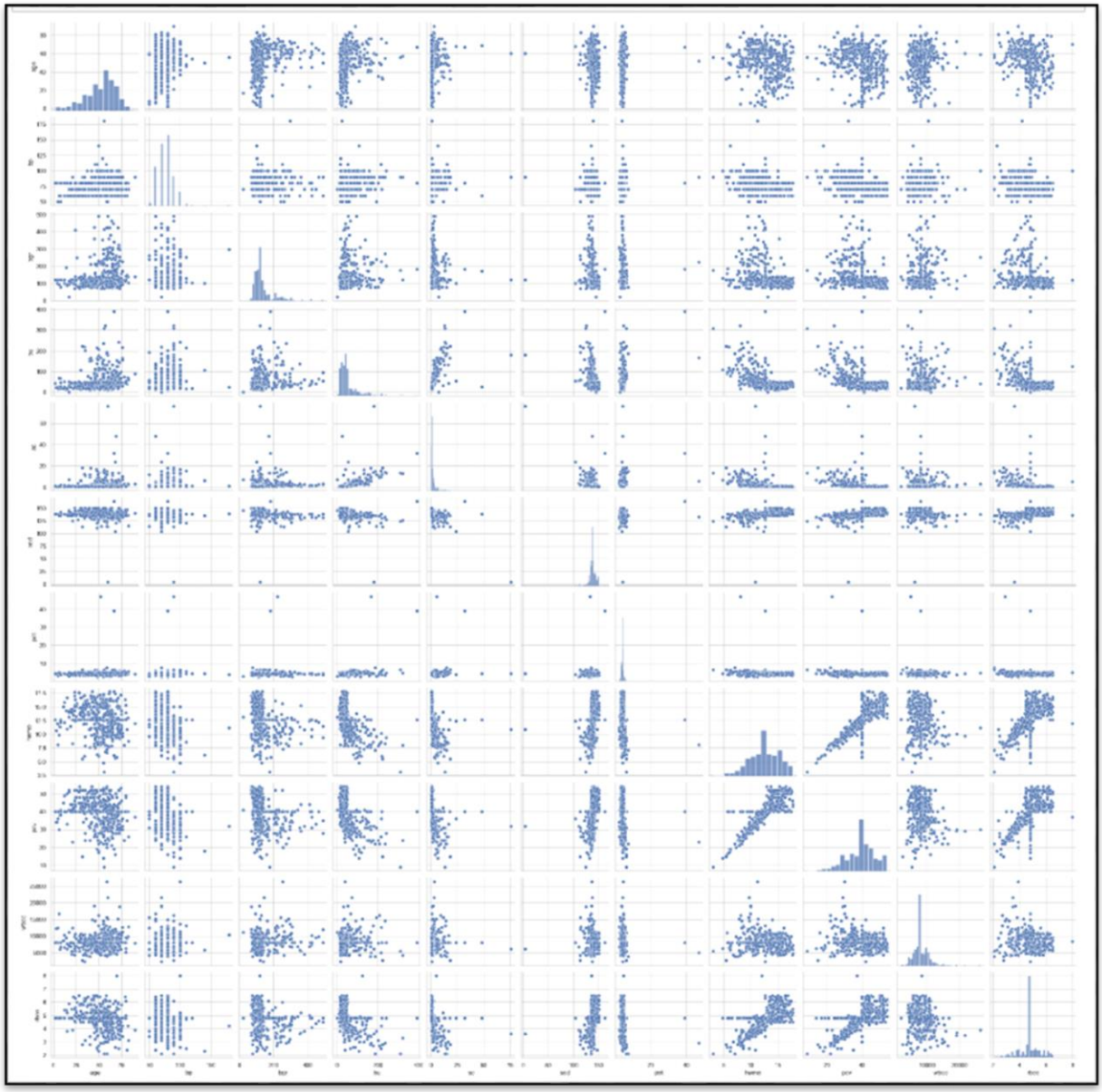


- Hemoglobin (hemo) and Packed Cell Volume (pcv) show a strong positive correlation (~0.90), indicating they are consistent indicators of each other's levels.
- Serum Creatinine (sc) and Sodium (sod) show a strong negative correlation (~-0.69), suggesting that as sc levels increase, sod levels decrease, potentially indicating kidney function impairment.

Pairplot Analysis:

Code:

```
In [24]: # Plot Pair plot
         sns.pairplot(df[df_numerical.columns])
         plt.show()
```



- Hemoglobin (hemo) and Packed Cell Volume (pcv) show a strong positive correlation, useful for diagnosing conditions like anemia or blood disorders.
- Age shows diverse patterns when plotted against various parameters, highlighting the complexity of age's interaction with physiological parameters and disease markers.
- Outliers are present in variables like bgr, Blood Urea (bu), and Serum Creatinine (sc), potentially indicating extreme cases or severe health conditions.

- Serum Creatinine (sc) and Blood Urea (bu) show a positive correlation, suggesting that their joint increase could signal kidney impairment or disease.

Target Class Distribution:

Code:

```
In [25]: # Target class distribution
         plt.figure(figsize=(6, 4))
         sns.countplot(x='class', data=df)
         plt.title('Distribution of Target Class')
         plt.show()
```

Target class distribution

The distribution of the target class in the dataset represents patients having chronic kidney disease (ckd) and those without (notckd). Specifically, there are **150 cases classified as ckd** and **250 cases** classified as **notckd**.

The dataset is imbalanced, with a larger number of notckd cases. This imbalance could influence the performance of a predictive model.

Feature Importance

Feature importance is a practice of assigning scores to the independent features for a machine learning model based on their influence in predicting the target variable, scores would be indicated which features are more significant in model's predictions and which features could potentially be ignored without loosing too much predictive power.

Information Gain (IG) is a measure helps to identify which features are the most informative for predicting the target variable & used with decision tree algorithms and is based on the concept of entropy from information theory.

```
# Splitting X and y for Predicting Feature importance
X = df.drop('class', axis=1)
y = df['class']
```

```
# Creating copy
X_label_encoded = X.copy()

# Select Categorical Columns
df_cat = X_label_encoded.select_dtypes(include=['object']).columns

# applying label encoding for categorical Columns
for col in df_cat:
    le = LabelEncoder()
    # Fit label encoder and return encoded labels
    X_label_encoded[col] = le.fit_transform(X[col].astype(str))
```

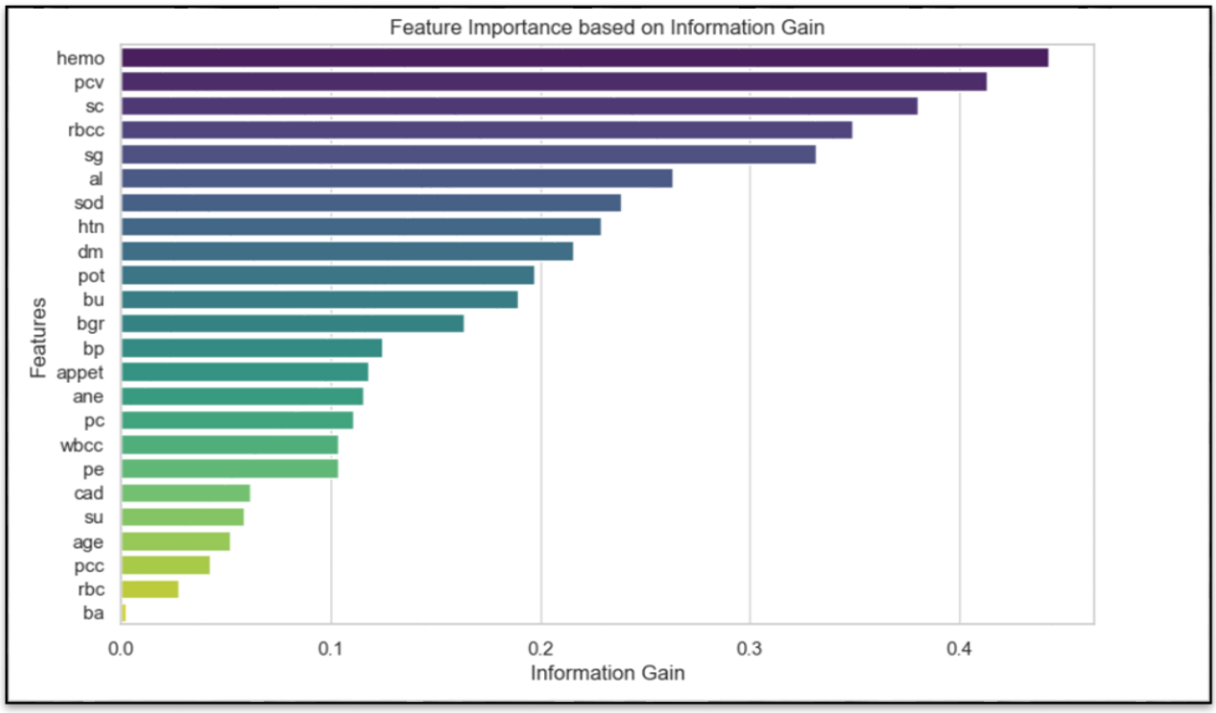
```
y = LabelEncoder().fit_transform(y)
```

Information Gain (IG) is a measure helps to identify which features are the most informative for predicting the target variable & used with decision tree algorithms and is based on the concept of entropy from information theory.

```
# Calculate Information Gain (mutual information) for each feature
info_gain = mutual_info_classif(X_label_encoded, y)
```

```
# Create a DataFrame for better visualization
info_gain_df = pd.DataFrame({'Feature': X_label_encoded.columns, 'Information Gain': info_gain}).sort_values(by='Information Gain
info_gain_df
```

Chart

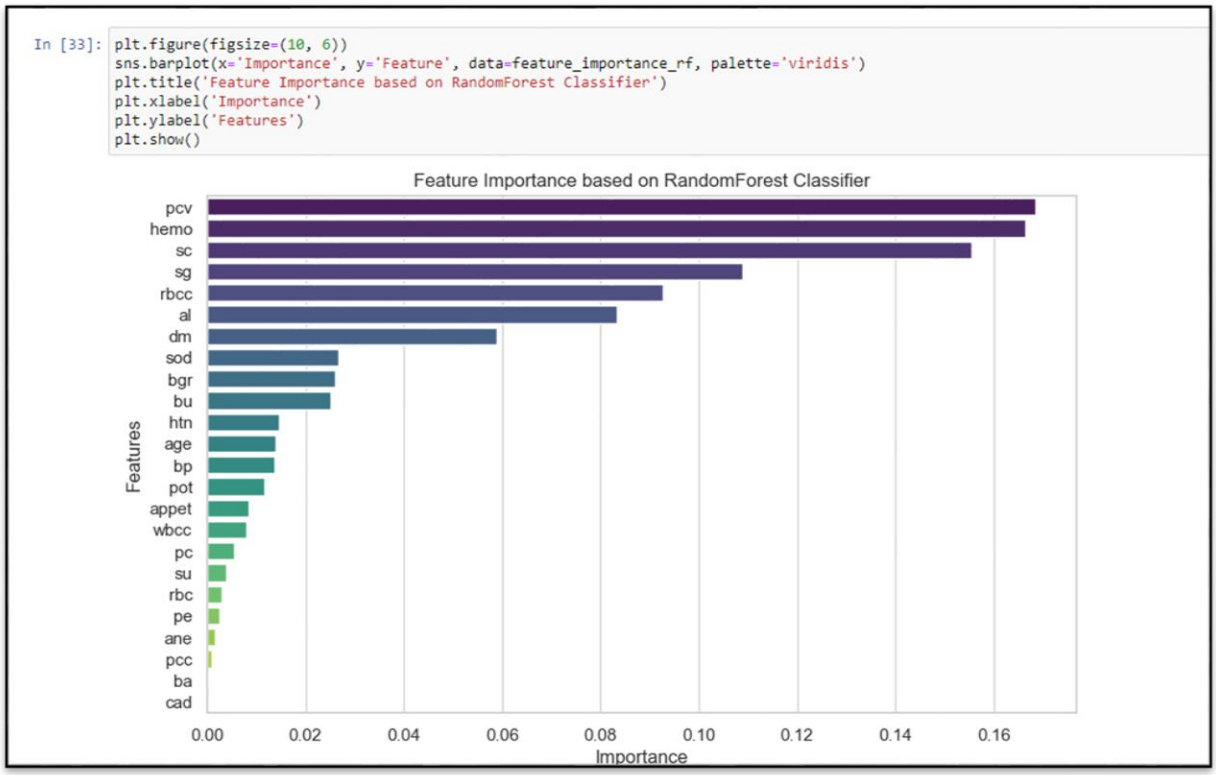


Random Forest uses a measure based on the decrease in node impurity & is Gini impurity or entropy in case of classification, and variance in case of regression. Every time a feature is used to split data, the impurity of the child nodes is calculated. The difference between the impurity of the parent node and the weighted impurity of the child nodes gives us the impurity decrease. The feature importance for a feature is calculated as the average impurity decrease for that feature across all trees in the forest.

```
In [32]: X_train, X_test, y_train, y_test = train_test_split(X_label_encoded, y, test_size=0.2, random_state=42)

         # Train a RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=100, random_state=42)
         rf.fit(X_train, y_train)

         # Get feature importance from the RandomForest model
         feature_importance_rf = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances_}).sort_values(by='Importance',
         feature_importance_rf
```

Chart

```
In [33]: plt.figure(figsize=(10, 6))
         sns.barplot(x='Importance', y='Feature', data=feature_importance_rf, palette='viridis')
         plt.title('Feature Importance based on RandomForest Classifier')
         plt.xlabel('Importance')
         plt.ylabel('Features')
         plt.show()
```



Feature Importance:

High Importance Features: Hemoglobin (hemo), Packed Cell Volume (pcv), Serum Creatinine (sc), Specific Gravity (sg), and Red Blood Cell Count (rbcc) are consistently ranked high in importance by both Random Forest and Information Gain, indicating its importance for prediction.

Moderately Important Features: Albumin (al), Diabetes Mellitus (dm), and Sodium (sod) shows moderate importance with slight ranking variations, suggesting their significant contribution for prediction.

Low Importance: Hypertension (htn), Blood Glucose Random (bgr), and Blood Urea (bu) show some ranking variance and are generally considered of moderate to low importance. Age, Blood Pressure (bp), and Potassium (pot) are ranked lower, indicating they might have less impact on the model.

No Importance: Pus Cell Clumps (pcc), Bacteria (ba), and Coronary Artery Disease (cad) consistently show little to no importance, suggesting it may not significantly influence the predictions and could be dropped without affecting performance significantly.

Dropping pcc & ba due to zero importance scores.

Train Test Split

Code:

```
In [34]: # Dropping the columns pcc and ba
         X = df.drop(columns=['class','pcc','ba'], axis=1)

In [35]: # Splitting 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,stratify=y)
         print("X_train shape:", X_train.shape)
         print("X_test shape:", X_test.shape)
         print("y_train shape:", y_train.shape)
         print("y_test shape:", y_test.shape)

         X_train shape: (320, 22)
         X_test shape: (80, 22)
         y_train shape: (320,)
         y_test shape: (80,)
```

Column Drop: Columns pcc and ba were dropped based on the feature importance.

Imbalanced Dataset: Stratification is used during data split to ensure balanced representation of each target class in the **train** and **test** sets, crucial for **avoiding bias** in classification tasks.

Split Ratio: Divided into **80%** for training and **20%** for testing.

Dataset Sizes:

X_train shape: (320, 22)
X_test shape: (80, 22)
y_train shape: (320,)
y_test shape: (80,)

Applying Standard scaling & Label Encoding

Create the custom method and function to apply fit for training and transform for test data

```
# Custom transformer for Label Encoding in multiple columns
class MultiColumnLabelEncoder(BaseEstimator, TransformerMixin):
    # fit method for training Datasets
    def fit(self, X, y=None):
        self.encoders = {column: LabelEncoder().fit(X[column]) for column in
X.select_dtypes(include=['object']).columns}
```

```
        return self  # Return the fitted object

    # transform method for test data
    def transform(self, X):
        X_encoded = X.copy()
        for column, encoder in self.encoders.items():
            X_encoded[column] = encoder.transform(X[column])
        return X_encoded  # Return the transformed DataFrame

# Getting numerical and categorical columns
numerical_cols = X_train.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_cols = X_train.select_dtypes(include=['object']).columns.tolist()

# Define the preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),  # Numerical features
        ('cat', MultiColumnLabelEncoder(), categorical_cols)  # Categorical features
    ], remainder='passthrough'  # This will keep all other columns untouched
)
```

Random forest Model:

In Task1, we have selected the **best model** is **Random Forest base model.** Hence I am training the Random Forest and below is the snippet of the code.

```
# Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(n_estimators=100, criterion='entropy',
random_state=10))
])
```

Experiment MLflow logging in Azure:

Code:

```
# Connect to Azure ML Workspace
experiment_name = "ckd-prediction-randomforest-basemodel"
experiment = Experiment(workspace=ws, name=experiment_name)

# Set the MLflow tracking URI to the Azure ML workspace
mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri())

# Set the 'GIT_PYTHON_REFRESH' environment variable to 'quiet'
os.environ["GIT_PYTHON_REFRESH"] = 'quiet'

# Set the experiment in MLflow to the Azure ML experiment
mlflow.set_experiment(experiment_name)
```

```
# Automatic logging of parameters, metrics, and models for scikit-learn training sessions
mlflow.sklearn.autolog()
```

1. **Connecting to Azure ML Workspace**: Initializes an Azure ML experiment using a pre-existing workspace and experiment name.
2. **Setting Environment Variable**: Sets the `GIT_PYTHON_REFRESH` environment variable to '**quiet**' to suppress GitPython warnings.
3. **Enabling MLflow Autologging for Scikit-learn**: Activates MLflow's autologging feature for scikit-learn to automatically capture training session information.
4. **Starting an MLflow Run**: Begins a new MLflow run to track individual executions of the machine learning code.

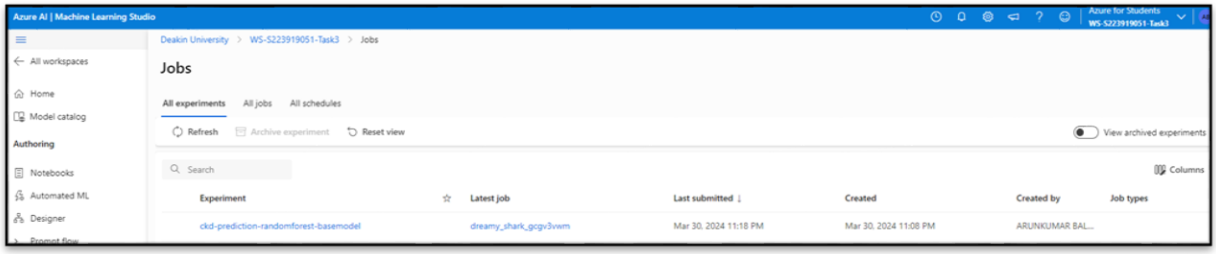
Base model fit evaluation and experiment logging

Code:

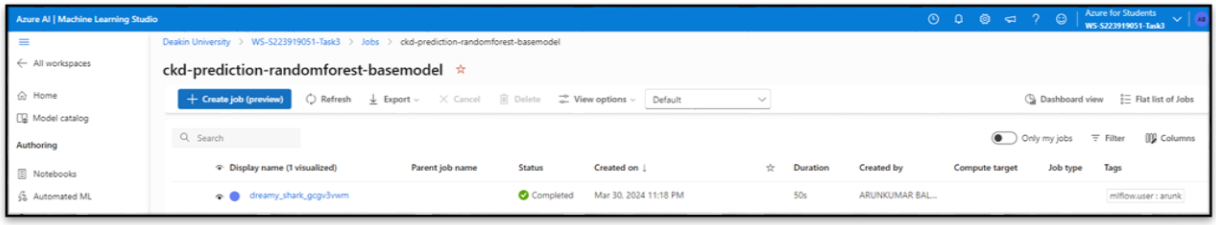
```
# Start an MLflow run
with mlflow.start_run():

    # fit the model
    rf = pipeline.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = pipeline.predict(X_test)
    y_proba = pipeline.predict_proba(X_test)[:, 1]

    # Calculate metrics
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba)

    # harmonic and geometric mean
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    sensitivity = tp / (tp + fn)  # Same as recall
    specificity = tn / (tn + fp)
    harmonic_mean = hmean(np.array([sensitivity, specificity]))
    geometric_mean = gmean(np.array([sensitivity, specificity]))

    # Log metrics
    mlflow.log_metric("ROC AUC", roc_auc)
    mlflow.log_metric("F1 Score", f1)
    mlflow.log_metric("Harmonic Mean", harmonic_mean)
    mlflow.log_metric("Geometric Mean", geometric_mean)
    cr = classification_report(y_test, y_pred, output_dict=True)
    mlflow.log_metric("accuracy", cr.pop("accuracy"))
    for class_or_avg, metrics_dict in cr.items():
        for metric, value in metrics_dict.items():
            mlflow.log_metric(f"{class_or_avg}_{metric}", value)
```

1. **Azure ML Workspace**: Initiates a connection and sets up an experiment for tracking model runs.
2. **Environment Variable**: Suppresses GitPython warnings for cleaner script output.
3. **MLflow Autologging**: Enables automatic logging of training details for scikit-learn models.
4. **MLflow Run**: Starts a tracking session, logging all model training details until the run ends.

Experiment posted in Azure:



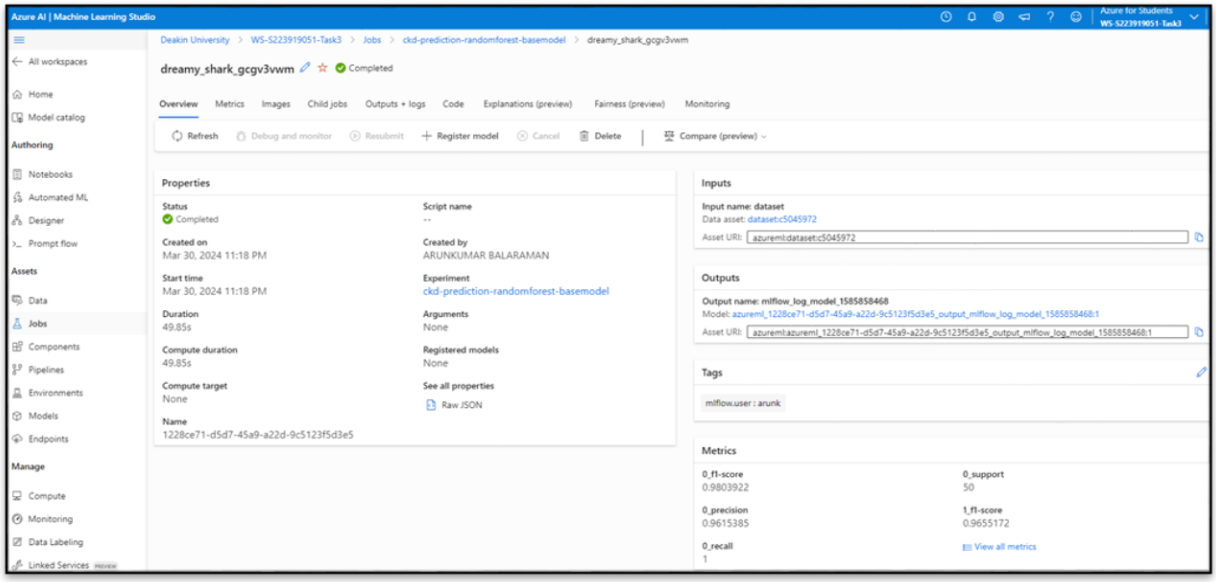
Job in experiment:



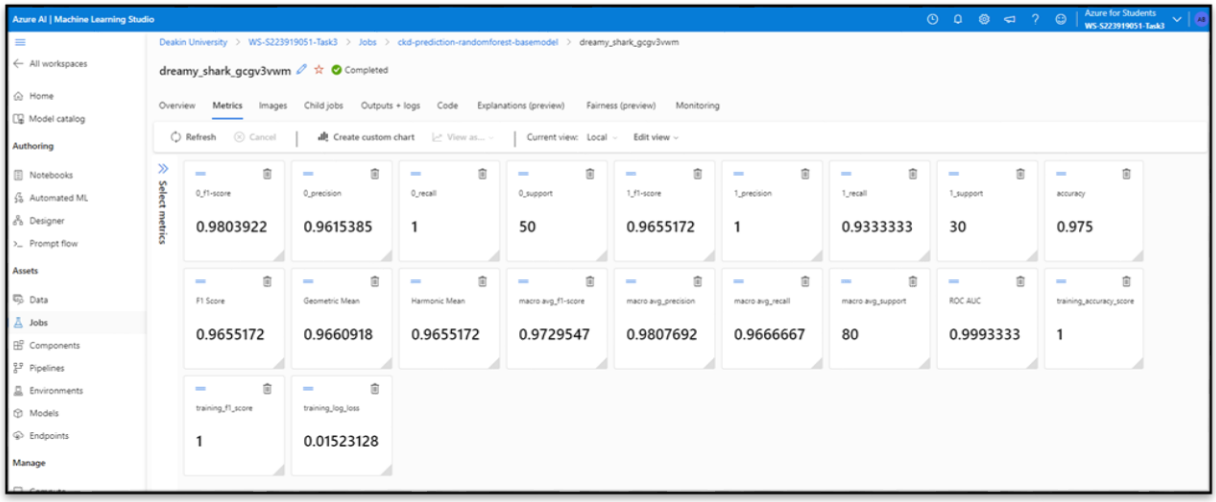
Classification Report in python SDK:

```
: print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        50
           1       1.00      0.93      0.97        30

    accuracy                           0.97        80
   macro avg       0.98      0.97      0.97        80
weighted avg       0.98      0.97      0.97        80
```

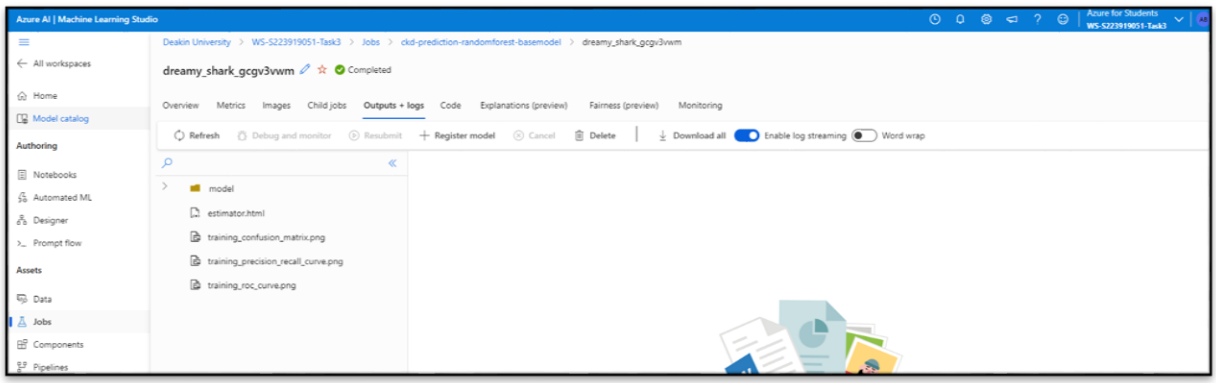
Job Details for the Job in ML Studio:



Metrics:



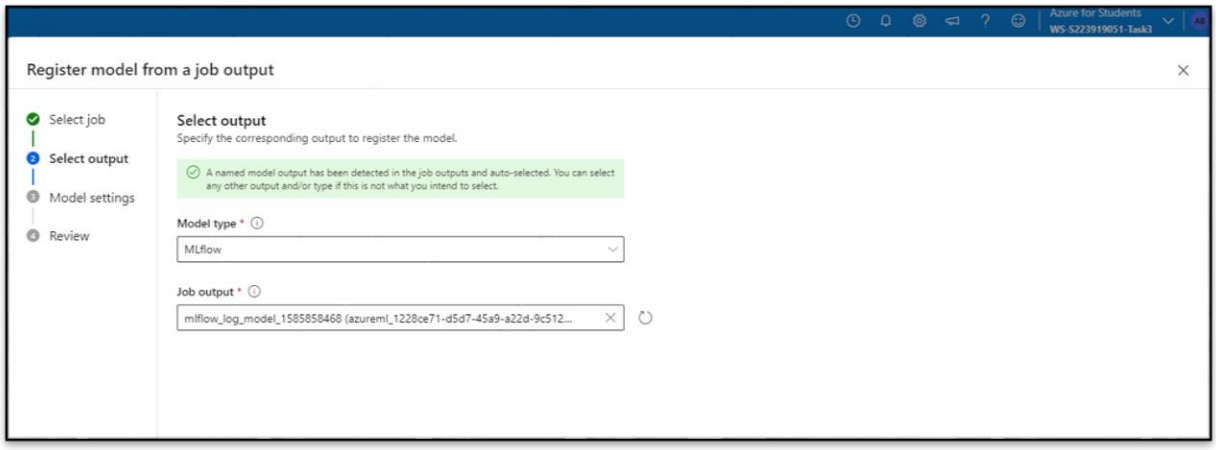
Logs & Output:



Register the model:

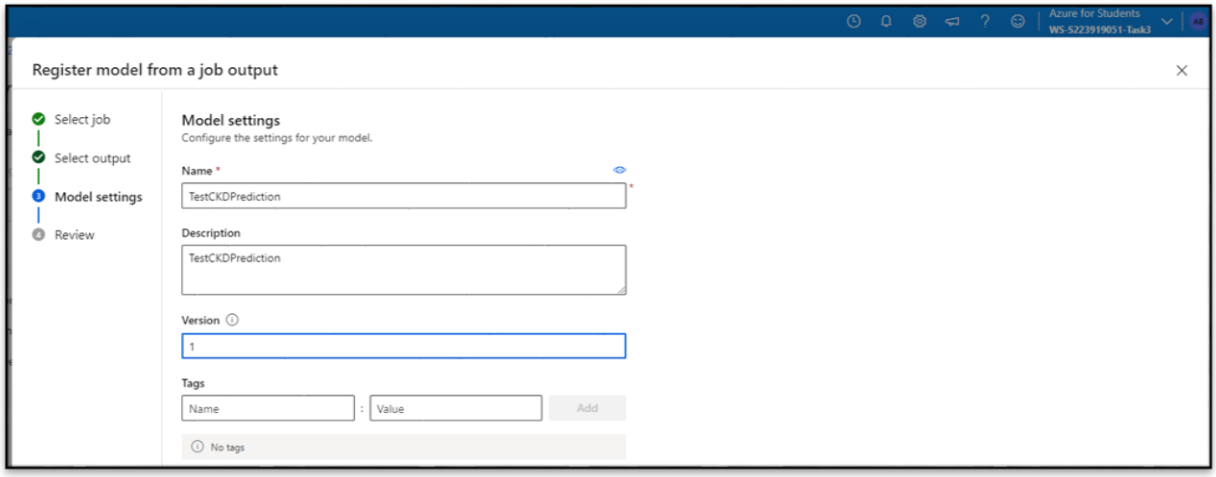
Registering the best performed model using Azure portal → Click on Register model → Select Model Type as **MLFlow** → Select the job output with the MLflow logged model in Python SDK.

Then click **Next**



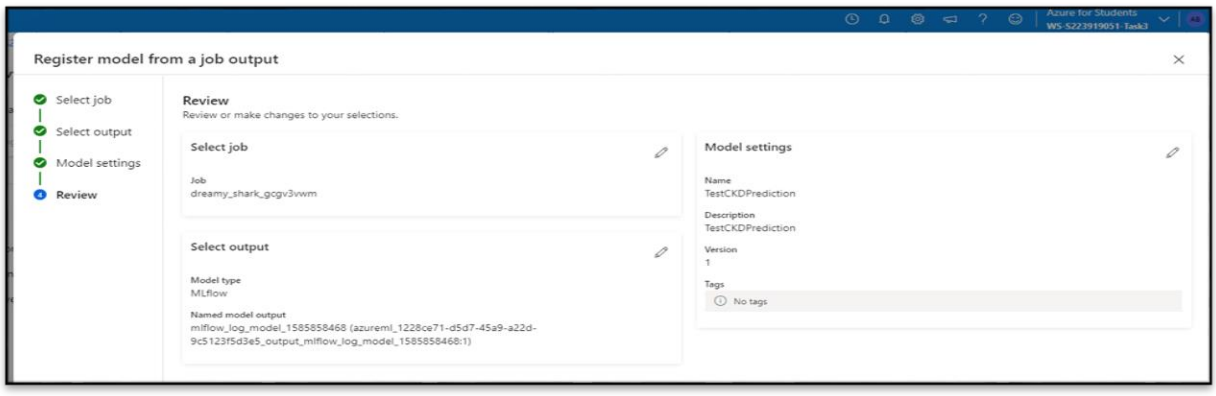
Model Settings:

Name: TestCKDPrediction
Description: TestCKDPrediction
Version: 1



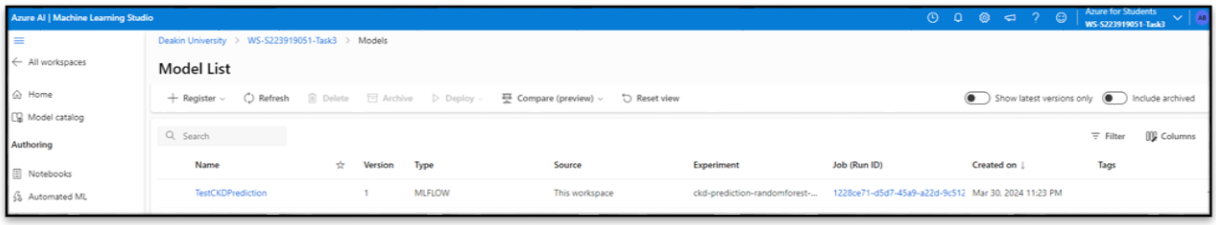
Then Click **Next:**

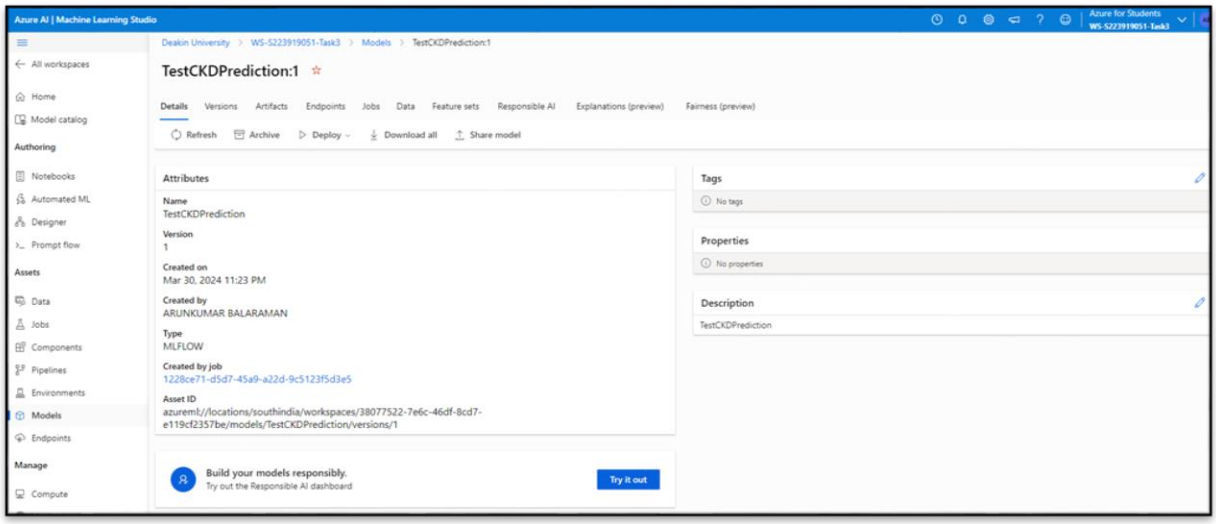
Review and Register:



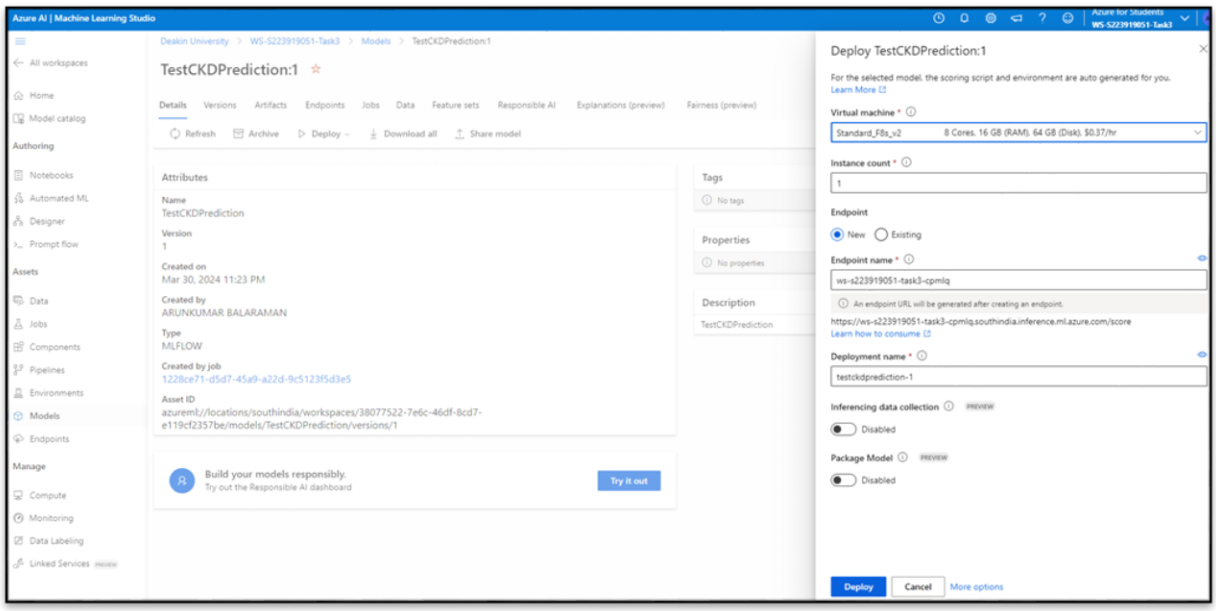
Models in Azure:

We have successfully registered the model type triton and with the experiment and job we created in Python SDK.

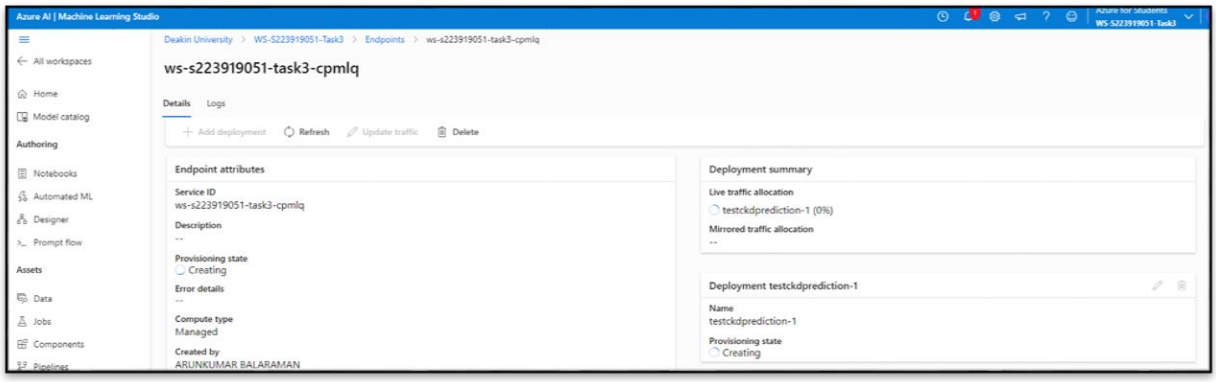




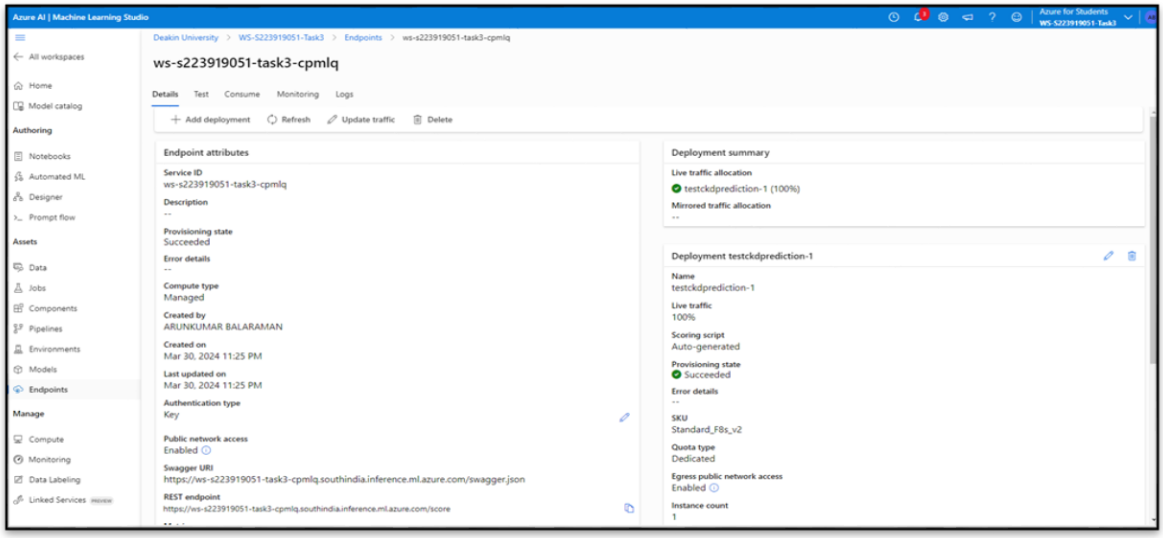
Deployment using Azure portal:



Deployment in Progress



Deployment & Endpoint Succeeded:



Test Deployment:

A sample JSON file has been created for testing purposes. It contains data at indices 0 and 1. Index 0 corresponds to 'Not CKD', represented by a test output of 0. Index 1 corresponds to 'CKD', represented by a test output of 1. The model has been successfully tested with this data.

Conclusion:

- Predictive analysis, machine learning models like Random Forest will help in early detection of CKD & Azure ML Studio would help in deploying the models to public.
- This approach enhances patient care by enabling timely interventions and personalized treatment plans.
- The study's success highlights machine learning's potential in medical diagnostics and its ability to handle complex datasets.

Next Steps:

- **Model Training**: A Random Forest model has been trained on CKD data. To ensure precision, it's beneficial to train and test additional models for a comprehensive evaluation and potential accuracy improvement.

- **Model Monitoring and CI/CD**: Continuous monitoring of the deployed model is crucial due to possible changes in data patterns. Implementing CI/CD pipelines can maintain model performance by automating testing and deployment processes. Automated monitoring systems can trigger alerts for timely interventions when model performance drops.

References

- Great Learning. (2024). **Introduction to Data Frames in pandas**. Available at: https://olympus.mygreatlearning.com/courses/97366/pages/4-dot-1-introduction-to-data-frames-in-pandas

- Great Learning. (2024). **Outliers, Missing, Censored, and Incorrect Data**. Available at: https://olympus.mygreatlearning.com/courses/97366/pages/4-dot-5-outliers-missing-censored-and-incorrect-data

- Great Learning Olympus. (2024). **Python SDK – Week2 Mentor Training**. Available at: https://olympus.mygreatlearning.com/mentorship_recordings/2288363

- Sruthi E R. (2024, Jan 03). **Understand Random Forest Algorithms With Examples (Updated 2024)** Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

- Brownlee, J. (No Date). **Metrics for Imbalanced Classification Books**. Available at: https://machinelearningmastery.com/imbalanced-classification-with-python/

- Goyal, P. (2018). **Feature Importance and Why It's Important**. Avialble at: https://towardsdatascience.com/feature-importance-and-why-its-important-c46d326e81d2

- Patel, K. (2018). **Why, How and When to Scale your Features**. Available at: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

- Brownlee, J. (October 16, 2019). **Information Gain and Mutual Information for Machine Learning. Machine Learning Mastery.** Available at: https://machinelearningmastery.com/information-gain-and-mutual-information/

- Sethi, A. (March 6, 2020). **One Hot Encoding vs. Label Encoding using Scikit-Learn**. Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

- Microsoft Developer (Jul 10, 2020) **How do you deploy a machine learning model as a web service within Azure** Available at: https://www.youtube.com/watch?v=n8h6_Expf38

- Microsoft Learn (no.date.) **Deploy a model as an online endpoint** Available at: https://learn.microsoft.com/en-us/azure/machine-learning/tutorial-deploy-model?view=azureml-api-2

- KonfHub Tech (Jun 3, 2020) **Using the Azure Machine Learning Python SDK to Train a PyTorch model at Scale by Henk Boelman** Available at: https://www.youtube.com/watch?v=5eORfEUp89Q&t=946s

- Microsoft Learn (no.date.) **What is an Azure Machine Learning workspace?** Available at: https://learn.microsoft.com/en-us/azure/machine-learning/tutorial-deploy-model?view=azureml-api-2