SEP 721 – Data Analytics, Machine Learning and AI on Cloud Platforms

Assignment 3: Azure Machine Learning

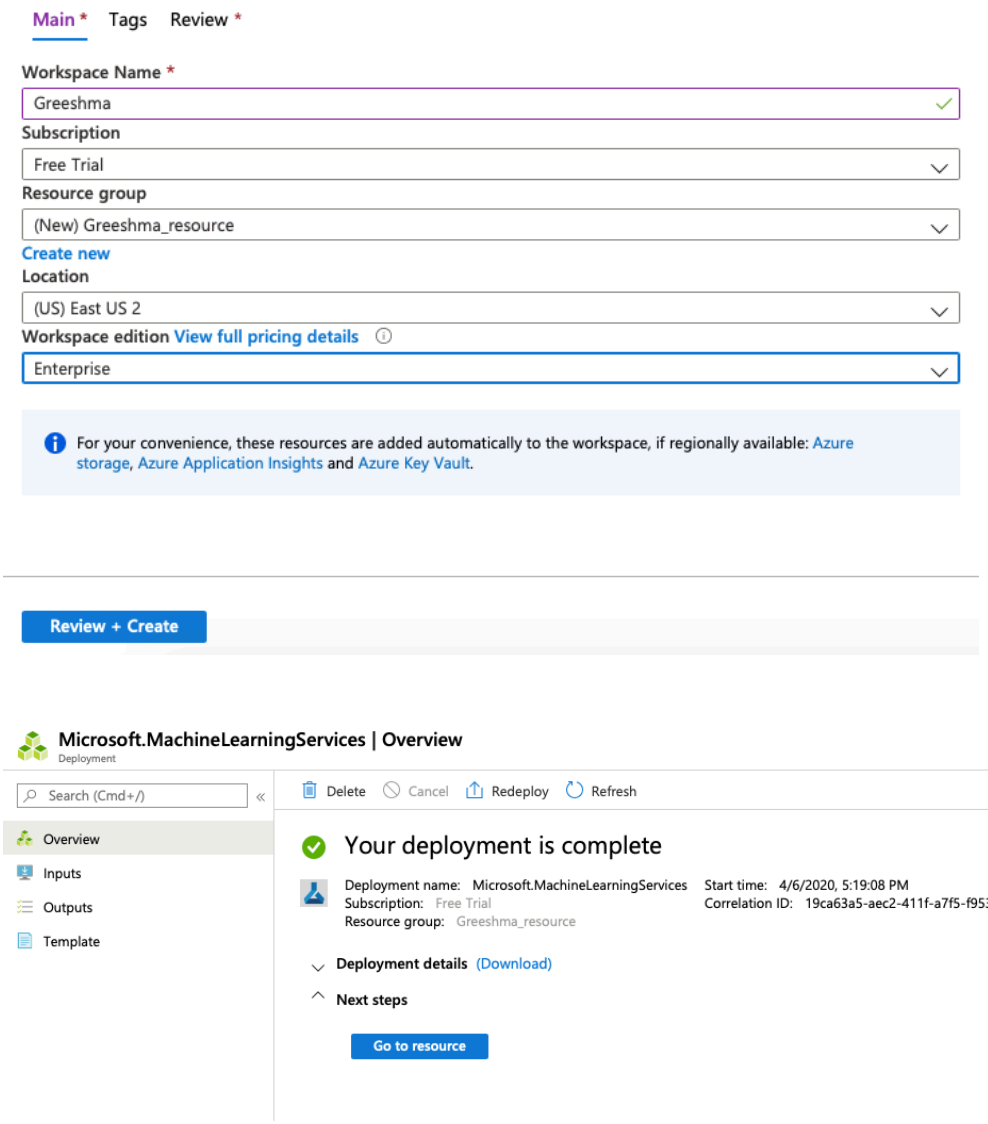
Submitted by,

Greeshma Gopal(gopalg)

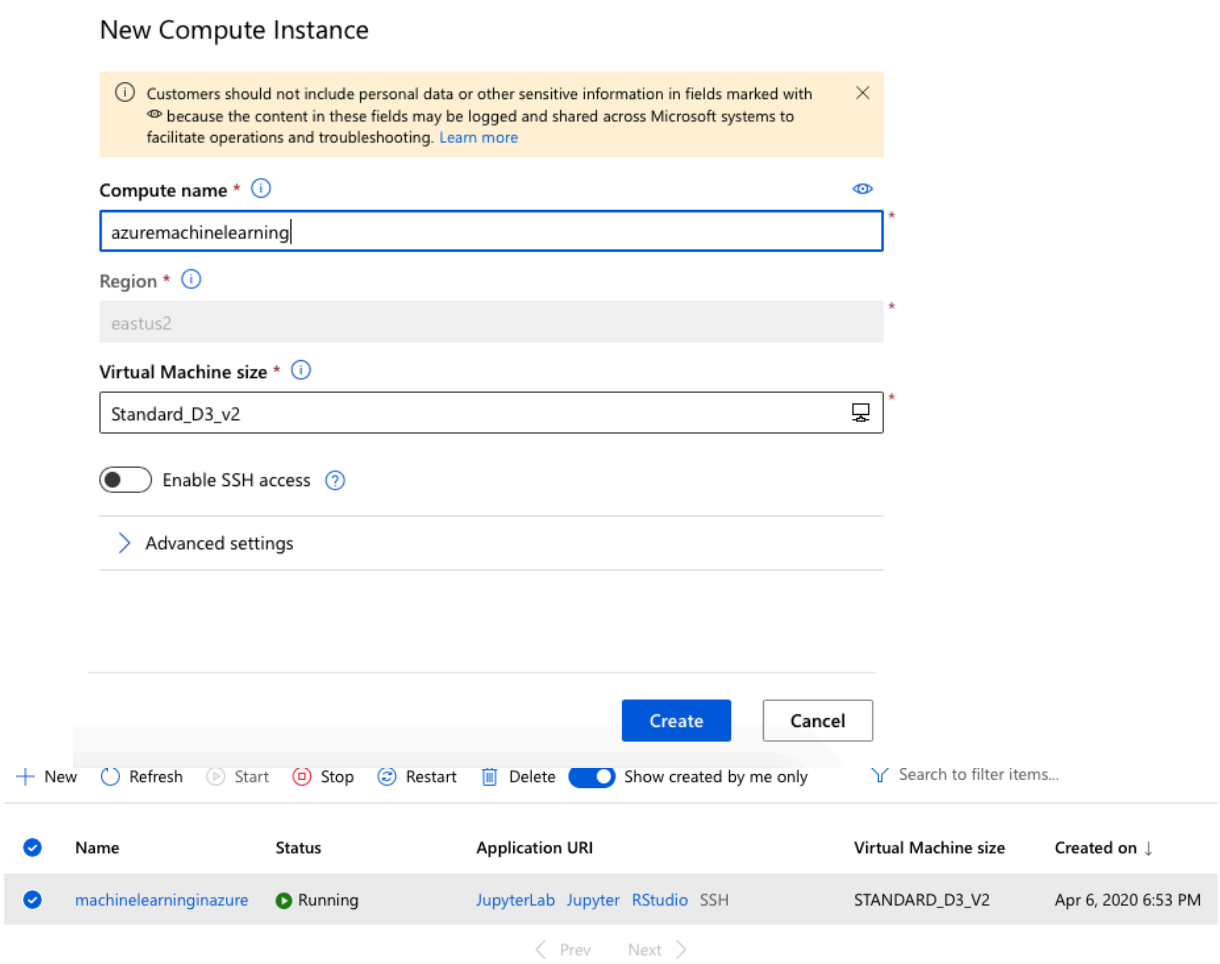
ID- 400245291

1. Creating Azure Machine learning

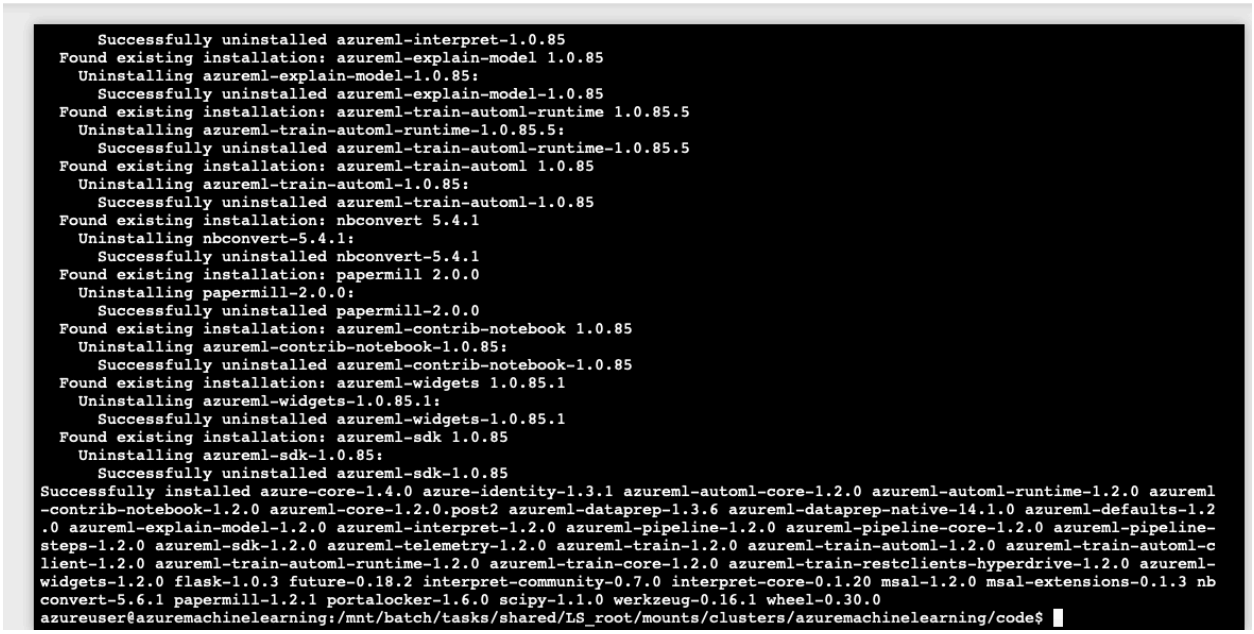
- Creating work space in azure cloud for machine learning



- Creating compute instance in Azure machine learning studio



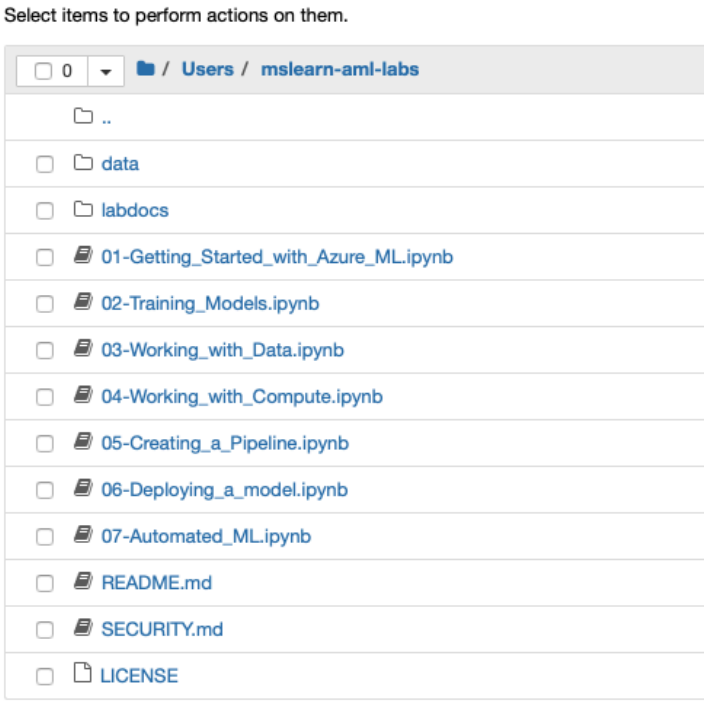
- Ensuring that the latest version of Azure machine learning is installed



- Cloning the github code

```
ttps://github.com/microsoftdocs/mslearn-aml-labs
Cloning into 'mslearn-aml-labs'...
remote: Enumerating objects: 147, done.
remote: Counting objects: 100% (147/147), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 147 (delta 89), reused 83 (delta 43), pack-reused 0
Receiving objects: 100% (147/147), 412.51 KiB | 0 bytes/s, done.
Resolving deltas: 100% (89/89), done.
Checking connectivity... done.
```

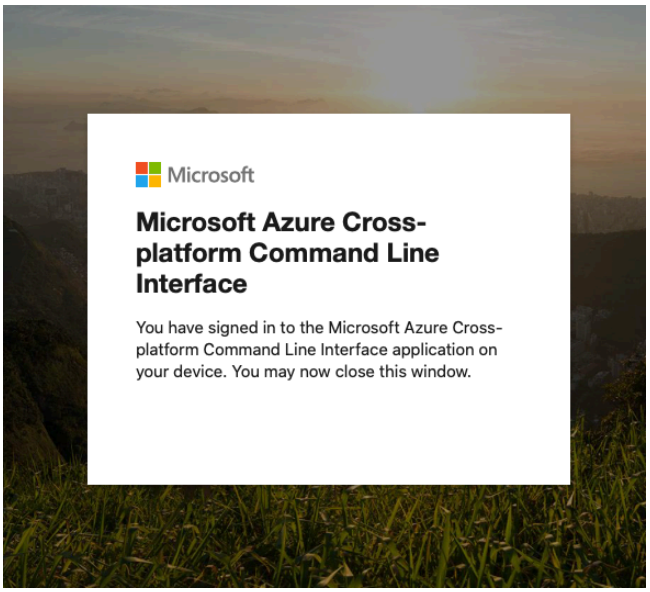
- The files have been cloned in the folder 'Users'



- Importing libraries and logging in the command line interface

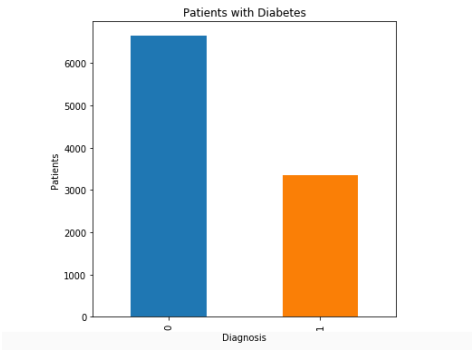
```
[2]: from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, "loaded")
```

```
Performing interactive authentication. Please follow the instructions
To sign in, use a web browser to open the page https://microsoft.com/d
thenticate.
Interactive authentication successfully completed.
greeshma1 loaded
```



- Run an experiment to explore the data, extracting statistics, visualizations, and data samples

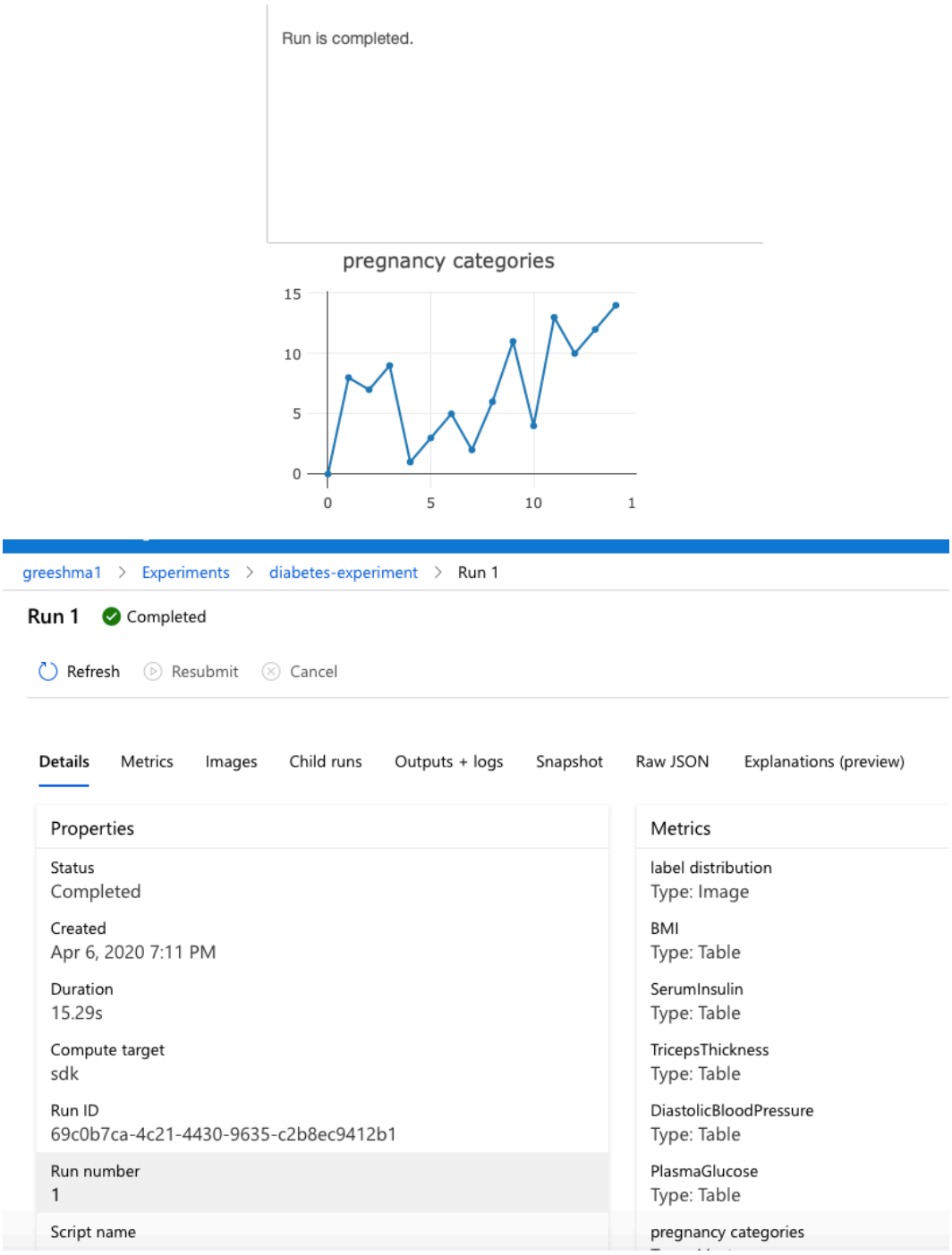
```
Starting experiment: diabetes-experiment
Analyzing 10000 rows of data
```



- Run object to get information about the run and its outputs

```
files = run.get_file_names()
print(json.dumps(files, indent=2))
```

```
learn-aml-labs', 'azureml.git.branch': 'master', 'mlflow.source.git.branch': 'master', 'azureml.git.commit': 'f3399
0a4fa96e44ca85bedb6961c50368c92d0f7', 'mlflow.source.git.commit': 'f33990a4fa96e44ca85bedb6961c50368c92d0f7', 'azur
eml.git.dirty': 'False', 'ContentSnapshotId': '96f1383d-29ac-488c-91c1-5b28bb120c30'}, 'inputDatasets': [], 'logFil
es': {}}
{
  "observations": 10000,
  "label distribution": "aml://artifactId/ExperimentRun/dcid.69c0b7ca-4c21-4430-9635-c2b8ec9412b1/label distributio
n_1586214726.png",
  "pregnancy categories": [
    0,
    8,
    7,
    9,
    1,
    3,
    5,
    2,
    6,
    11,
```



- Creating folder for the experiment files, and copy the data into it

```
os.makedirs('outputs', exist_ok=True)
data.sample(100).to_csv("outputs/sample.csv", index=False, header=True)

# Complete the run
run.complete()
```

```
Writing diabetes-experiment-files/diabetes_experiment.py
```

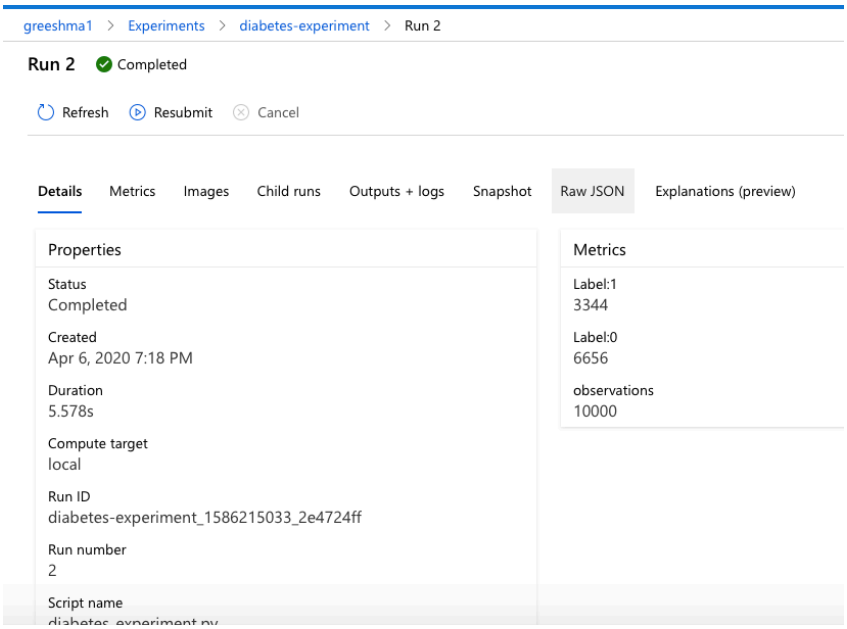
- Creating conda environment to run the experiment 2

	33_2e4724ff
Arguments	N/A
Label:0	6656
Label:1	3344
observations	10000

click to scroll output; double click to hide

logs/azureml/8590_azureml.log Auto-switch

```
2020-04-06
23:19:00,390|azureml._history.utils.context_managers.FileWatcher.UploadQueue|DEBUG|Finished
calling finish
2020-04-06
23:19:00,390|azureml._history.utils.context_managers.FileWatcher|DEBUG|FileWatcher task queue
finished, exiting
2020-04-06 23:19:00,391|azureml._history.utils.context_managers.FileWatcher.10d8c378-d009-
4b29-b445-56dc4126c6e3|DEBUG|Using basic handler - no exception handling

Run is completed.
```



```
Out[8]: {'runId': 'diabetes-experiment_1586215033_2e4724ff',
         'target': 'local',
         'status': 'Finalizing',
         'startTimeUtc': '2020-04-06T23:18:56.833195Z',
         'properties': {'_azureml.ComputeTargetType': 'local',
          'ContentSnapshotId': '4b336503-4ab2-4618-b5f1-5756e4b1e48f',
          'azureml.git.repository_uri': 'https://github.com/microsoftdocs/mslearn-aml-labs',
          'mlflow.source.git.repoURL': 'https://github.com/microsoftdocs/mslearn-aml-labs',
          'azureml.git.branch': 'master',
          'mlflow.source.git.branch': 'master',
          'azureml.git.commit': 'f33990a4fa96e44ca85bedb6961c50368c92d0f7',
          'mlflow.source.git.commit': 'f33990a4fa96e44ca85bedb6961c50368c92d0f7',
          'azureml.git.dirty': 'True'},
         'inputDatasets': [],
         'runDefinition': {'script': 'diabetes_experiment.py',
          'useAbsolutePath': False,
          'arguments': [],
          'sourceDirectoryDataStore': None,
          'framework': 'Python',
          'communicator': 'None',
          'target': 'local',
          'dataReferences': {},
          'data': {},
          'jobName': None,
          'maxRunDurationSeconds': None,
          'nodeCount': 1,
          'environment': {'name': 'Experiment diabetes-experiment Environment',
           'version': 'Autosave_2020-04-06T23:17:14Z_71f7f214',
```

- Retrieving the metrics and files it generated

```
[9]: # Get logged metrics
     metrics = run.get_metrics()
     for key in metrics.keys():
             print(key, metrics.get(key))
     print('\n')
     for file in run.get_file_names():
         print(file)
```

```
observations 10000
Label:0 6656
Label:1 3344


azureml-logs/60_control_log.txt
azureml-logs/70_driver_log.txt
logs/azureml/8590_azureml.log
outputs/sample.csv
```

- Retrieving an experiment by name from the workspace and iterate through its runs using the SDK

```
0]: from azureml.core import Experiment, Run

    diabetes_experiment = ws.experiments['diabetes-experiment']
    for logged_run in diabetes_experiment.get_runs():
        print('Run ID:', logged_run.id)
        metrics = logged_run.get_metrics()
        for key in metrics.keys():
            print('-', key, metrics.get(key))
```

```
Run ID: diabetes-experiment_1586215033_2e4724ff
- observations 10000
- Label:0 6656
- Label:1 3344
Run ID: 69c0b7ca-4c21-4430-9635-c2b8ec9412b1
- observations 10000
- label distribution aml://artifactId/ExperimentRun/dcid.69c0b7ca-4c21-4430-9635-c2b8ec9412b1/label distribution_15
86214726.png
- pregnancy categories [0, 8, 7, 9, 1, 3, 5, 2, 6, 11, 4, 13, 10, 12, 14]
- PlasmaGlucose {'stat': ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max'], 'value': [10000.0, 107.8502,
31.920909360565563, 44, 84, 105, 129, 192]}
- DiastolicBloodPressure {'stat': ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max'], 'value': [10000.0, 7
1.2075, 16.801478289640706, 24, 58, 72, 85, 117]}
- TricepsThickness {'stat': ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max'], 'value': [10000.0, 28.8176
, 14.506480415228332, 7, 15, 31, 41, 92]}
- SerumInsulin {'stat': ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max'], 'value': [10000.0, 139.2436, 1
33.77791937465278, 14, 39, 85, 197, 796]}
- BMI {'stat': ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max'], 'value': [10000.0, 31.56702174359113, 9
.804365693559113, 18.20080735, 21.247426835, 31.922420785, 39.3289214475, 56.03462763]}
```

2. Training Models

- Connecting with the work space

In [1]:
```
import azureml.core
from azureml.core import Workspace

# Load the workspace from the saved config file
ws = Workspace.from_config()
print('Ready to use Azure ML {} to work with {}'.format(azureml.core.VERSION, ws.name))
```

```
Ready to use Azure ML 1.2.0 to work with greeshma1
```

- Creating folders and files for the script

In [2]:
```
import os, shutil

# Create a folder for the experiment files
training_folder = 'diabetes-training'
os.makedirs(training_folder, exist_ok=True)

# Copy the data file into the experiment folder
shutil.copy('data/diabetes.csv', os.path.join(training_folder, "diabetes.csv"))
```

Out[2]:
```
'diabetes-training/diabetes.csv'
```

- Creating training script and saving in folder

```
# calculate AUC
y_scores = model.predict_proba(X_test)
auc = roc_auc_score(y_test,y_scores[:,1])
print('AUC: ' + str(auc))
run.log('AUC', np.float(auc))

# Save the trained model in the outputs folder
os.makedirs('outputs', exist_ok=True)
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

```
Writing diabetes-training/diabetes_training.py
```

- Using generic estimator to run the training experiment. The default environment for this estimator does not include the scikit-learn package, so this has to be added explicitly to the configuration. The conda environment is built on-demand the first time the estimator is used, and cached for future runs that use the same configuration.

Run Properties

Status	Completed
Start Time	2020-04-06 9:51:32 PM
Duration	0:00:16
Run Id	diabetes-training_1586224290_f8d320c3
Arguments	N/A
Accuracy	0.774
AUC	0.8484929598487486
Regularization Rate	0.01

- Getting the metrics

```
In [9]: # Get logged metrics
        metrics = run.get_metrics()
        for key in metrics.keys():
                print(key, metrics.get(key))
        print('\n')
        for file in run.get_file_names():
            print(file)
```

```
Regularization Rate 0.01
Accuracy 0.774
AUC 0.8484929598487486


azureml-logs/60_control_log.txt
azureml-logs/70_driver_log.txt
logs/azureml/8_azureml.log
outputs/diabetes_model.pkl
```

- Registering the model to track versions and fetch the model later

In [10]:
```
from azureml.core import Model

# Register the model
run.register_model(model_path='outputs/diabetes_model.pkl', model_name='diabetes_model',
                   tags={'Training context':'Estimator'},
                   properties={'AUC': run.get_metrics()['AUC'], 'Accuracy': run.get_metrics()['Accuracy']})

# List registered models
for model in Model.list(ws):
    print(model.name, 'version:', model.version)
    for tag_name in model.tags:
        tag = model.tags[tag_name]
        print ('\t',tag_name, ':', tag)
    for prop_name in model.properties:
        prop = model.properties[prop_name]
        print ('\t',prop_name, ':', prop)
    print('\n')
```

```
diabetes_model version: 1
	 Training context : Estimator
	 AUC : 0.8484929598487486
	 Accuracy : 0.774
```

- Creating folder for parameterized training script

In [11]:
```
import os, shutil

# Create a folder for the experiment files
training_folder = 'diabetes-training-params'
os.makedirs(training_folder, exist_ok=True)

# Copy the data file into the experiment folder
shutil.copy('data/diabetes.csv', os.path.join(training_folder, "diabetes.csv"))
```

Out[11]:
```
'diabetes-training-params/diabetes.csv'
```

```
# calculate AUC
y_scores = model.predict_proba(X_test)
auc = roc_auc_score(y_test,y_scores[:,1])
print('AUC: ' + str(auc))
run.log('AUC', np.float(auc))

os.makedirs('outputs', exist_ok=True)
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

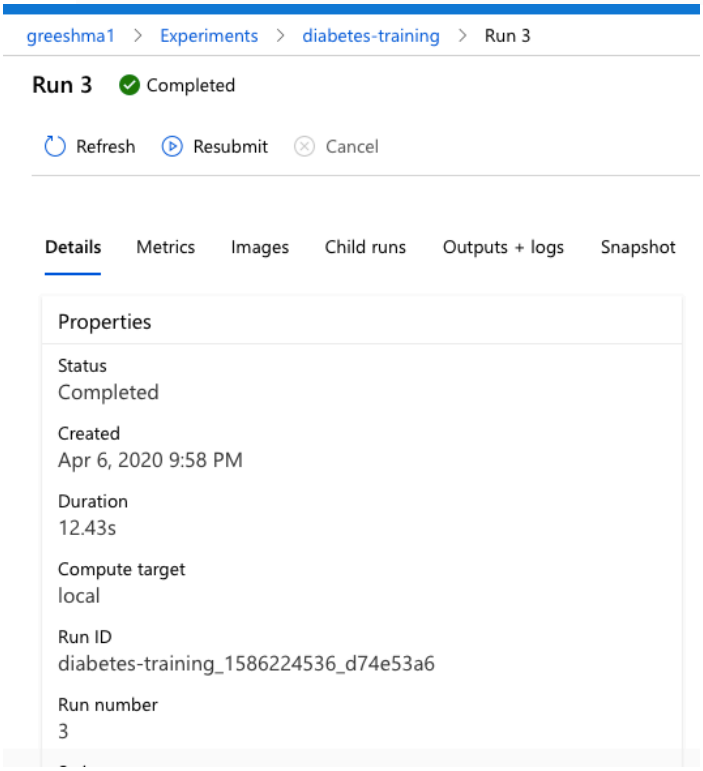
```
Writing diabetes-training-params/diabetes_training.py
```

- Running using framework specific estimator

```
run.wait_for_completion()
```

Run Properties

Status	Completed
Start Time	2020-04-06 9:55:37 PM
Duration	0:02:39
Run Id	diabetes-training_1586224536_d74e53a6
Arguments	N/A
Accuracy	0.7736666666666666
AUC	0.8483904671874223



- Fetching the metrics

In [14]:
```
# Get logged metrics
metrics = run.get_metrics()
for key in metrics.keys():
        print(key, metrics.get(key))
print('\n')
for file in run.get_file_names():
    print(file)
```

```
Regularization Rate 0.1
Accuracy 0.7736666666666666
AUC 0.8483904671874223


azureml-logs/60_control_log.txt
azureml-logs/70_driver_log.txt
logs/azureml/8_azureml.log
outputs/diabetes_model.pkl
```

- Registering the new version of the model

```
                    tags={'Training context':'Parameterized SKLea
                    properties={'AUC': run.get_metrics()['AUC'],

# List registered models
for model in Model.list(ws):
    print(model.name, 'version:', model.version)
    for tag_name in model.tags:
        tag = model.tags[tag_name]
        print ('\t',tag_name, ':', tag)
    for prop_name in model.properties:
        prop = model.properties[prop_name]
        print ('\t',prop_name, ':', prop)
    print('\n')
```

```
diabetes_model version: 2
         Training context : Parameterized SKLearn Estimator
         AUC : 0.8483904671874223
         Accuracy : 0.7736666666666666


diabetes_model version: 1
         Training context : Estimator
         AUC : 0.8484929598487486
         Accuracy : 0.774
```

3. Working with Data

- Connecting to the work space

In [1]:
```
import azureml.core
from azureml.core import Workspace

# Load the workspace from the saved config file
ws = Workspace.from_config()
print('Ready to use Azure ML {} to work with {}'.format(azurem
```

```
Ready to use Azure ML 1.2.0 to work with greeshma1
```

- Determining the data stores of the workspace

[2]:
```
# Get the default datastore
default_ds = ws.get_default_datastore()

# Enumerate all datastores, indicating which is the default
for ds_name in ws.datastores:
    print(ds_name, "- Default =", ds_name == default_ds.name)
```

```
workspaceblobstore - Default = True
workspacefilestore - Default = False
```

- Uploading data into the datastore

```
                        show_progress=True)
```

```
Uploading an estimated of 2 files
Uploading ./data/diabetes.csv
Uploading ./data/diabetes2.csv
Uploaded ./data/diabetes.csv, 1 files out of an estimated total of 2
Uploaded ./data/diabetes2.csv, 2 files out of an estimated total of 2
Uploaded 2 files
```

[3]:
```
$AZUREML_DATAREFERENCE_468d27663e024b29b89a1391b9228327
```

- Train the model from a datastore and create folder for the same

```
]: data_ref = default_ds.path('diabetes-data').as_download(path_on_compute='diabetes_data')
print(data_ref)
```

```
$AZUREML_DATAREFERENCE_d566947648754b6fa996dff97238918d
```

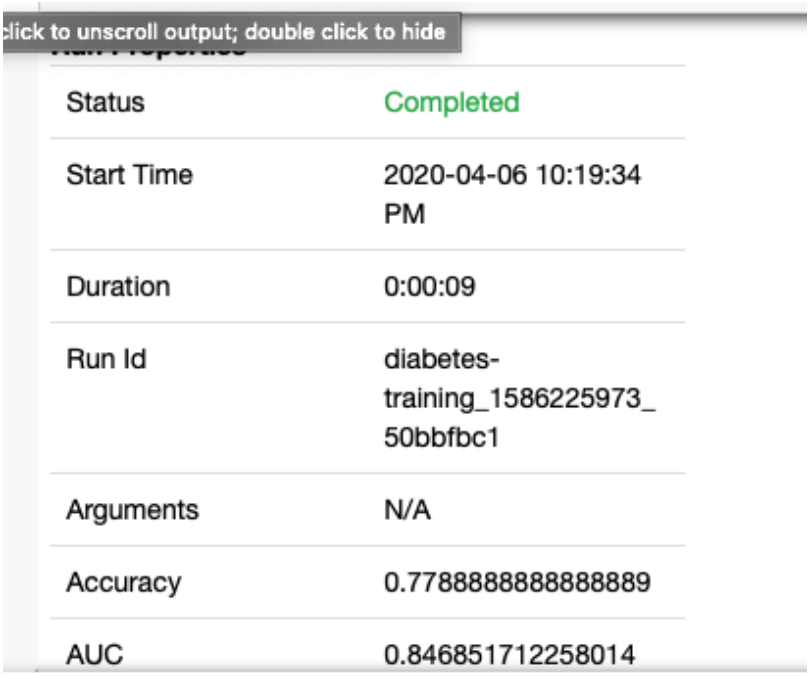
```
experiment_folder = 'diabetes_training_from_datastore'
os.makedirs(experiment_folder, exist_ok=True)
print(experiment_folder, 'folder created.')
```

```
diabetes_training_from_datastore folder created.
```

```
# note file saved in the outputs folder is automatically uploaded i
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

```
Writing diabetes_training_from_datastore/diabetes_training.py
```

- Setting up the script parameters to pass the file reference when we run the experiment



- Creating a dataset from the diabetes data you uploaded to the datastore, and view the first 20 records. In this case, the data is in a structured format in a CSV file, hence using tabular dataset

```
tab_data_set.take(20).to_pandas_dataframe()
```

]:

	PatientID	Pregnancies	PlasmaGlucose	DiastolicBloodPressure	TricepsThickness	SerumInsulin	BMI	DiabetesPedigree	Age	Diabetic
0	1354778	0	171	80	34	23	43.509726	1.213191	21	0
1	1147438	8	92	93	47	36	21.240576	0.158365	23	0
2	1640031	7	115	47	52	35	41.511523	0.079019	23	0
3	1883350	9	103	78	25	304	29.582192	1.282870	43	1
4	1424119	1	85	59	27	35	42.604536	0.549542	22	0
	click to unscroll output; double click to hide		82	92	9	253	19.724160	0.103424	26	0
6	1660149	0	133	47	19	227	21.941357	0.174160	21	0
7	1458769	0	67	87	43	36	18.277723	0.236165	26	0
8	1201647	8	80	95	33	24	26.624929	0.443947	53	1
9	1403912	1	72	31	40	42	36.889576	0.103944	26	0
10	1943830	1	88	86	11	58	43.225041	0.230285	22	0

- Creating file dataset, which creates a list of file paths in a virtual mount point, to read the data in the files.

In [9]:
```
#Create a file dataset from the path on the datastore (this may take a short while
file_data_set = Dataset.File.from_files(path=(default_ds, 'diabetes-data/*.csv'))

# Get the files in the dataset
for file_path in file_data_set.to_path():
    print(file_path)
```

```
/diabetes.csv
/diabetes2.csv
```

- Registering the data stores

```
# Register the file dataset
try:
    file_data_set = file_data_set.register(workspace=ws,
                                           name='diabetes file dataset',
                                           description='diabetes files',
                                           tags = {'format':'CSV'},
                                           create_new_version=True)
except Exception as ex:
    print(ex)

print('Datasets registered')
```

```
Datasets registered
```

- Printing the list of data sets in the workspace

[11]:
```
print("Datasets:")
for dataset_name in list(ws.datasets.keys()):
    dataset = Dataset.get_by_name(ws, dataset_name)
    print("\t", dataset.name, 'version', dataset.version)
```

```
Datasets:
         diabetes file dataset version 1
         diabetes dataset version 1
```

Train a Model from a tabular Dataset

- Create folder for the data set

2]:
```
import os

# Create a folder for the experiment files
experiment_folder = 'diabetes_training_from_tab_dataset'
os.makedirs(experiment_folder, exist_ok=True)
print(experiment_folder, 'folder created')
```

```
diabetes_training_from_tab_dataset folder created
```

```
os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploaded into experiment record
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

```
Writing diabetes_training_from_tab_dataset/diabetes_training.py
```

- Running the experiment

Run Properties	
Status	Completed
Start Time	2020-04-06 10:29:20 PM
Duration	0:03:18
Run Id	diabetes-training_1586226559_284debac
Arguments	N/A
Accuracy	0.7893333333333333
AUC	0.8568632924585982

Train a Model from a File Dataset

- Creating folders for training

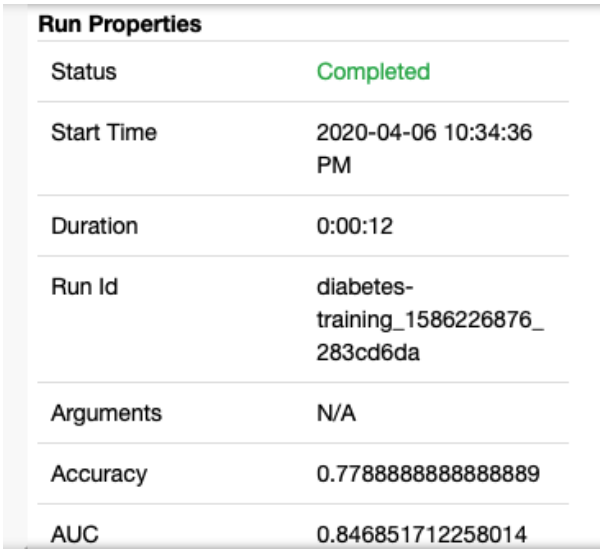
```
experiment_folder = 'diabetes_training_from_file_data
os.makedirs(experiment_folder, exist_ok=True)
print(experiment_folder, 'folder created')
```

```
diabetes_training_from_file_dataset folder created
```

```
os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploaded into experiment
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

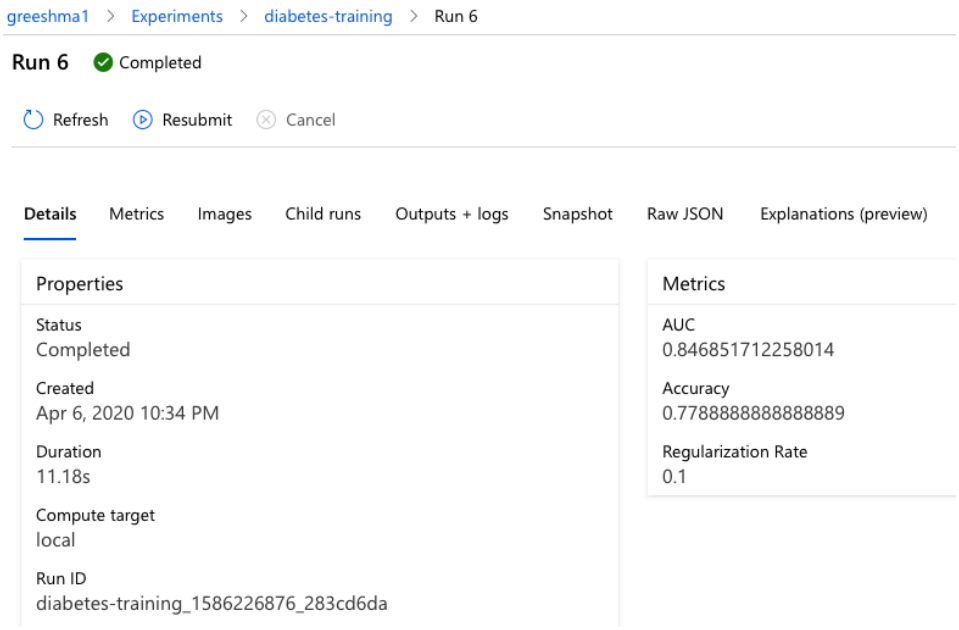
```
Writing diabetes_training_from_file_dataset/diabetes_training.py
```

- Running the experiment



Run Properties	
Status	Completed
Start Time	2020-04-06 10:34:36 PM
Duration	0:00:12
Run Id	diabetes-training_1586226876_283cd6da
Arguments	N/A
Accuracy	0.7788888888888889
AUC	0.846851712258014

- Viewing the status in Azure machine learning studio



4. Working with Compute

- Connecting to the work space

```
    except Exception as ex:
        print(ex)
else:
    print('Dataset already registered.')
```

```
Dataset already registered.
```

- Creating folders for the logistic experiment

```
os.makedirs(experiment_folder, exist_ok=True)
print(experiment_folder, 'folder created')
```

```
diabetes_training_logistic folder created
```

```
auc = roc_auc_score(y_test,y_scores[:,1])
print('AUC: ' + str(auc))
run.log('AUC', np.float(auc))

os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploade
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

```
Writing diabetes_training_logistic/diabetes_training.py
```

- Defining the environment and adding packages by using **conda** or **pip**, to ensure the experiment has access to all the libraries it requires.

```
# Create a Python environment for the experiment
diabetes_env = Environment("diabetes-experiment-env")
diabetes_env.python.user_managed_dependencies = False # Let Azure ML manage dependencies
diabetes_env.docker.enabled = True # Use a docker container

# Create a set of package dependencies (conda or pip as required)
diabetes_packages = CondaDependencies.create(conda_packages=['scikit-learn'],
                                             pip_packages=['azureml-defaults', 'azureml-dataprep[pandas]'])

# Add the dependencies to the environment
diabetes_env.python.conda_dependencies = diabetes_packages

print(diabetes_env.name, 'defined.')
```

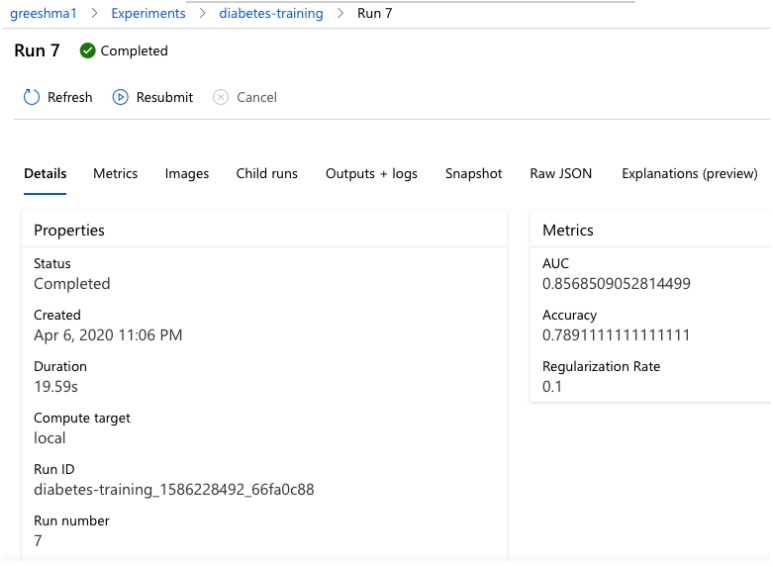
```
diabetes-experiment-env defined.
```

- Assigning the environment created to a generic estimator, and submitting an experiment. Conda environment will also be created

```
RunDetails(run).show()
run.wait_for_completion()
```

Run Properties

Status	Completed
Start Time	2020-04-06 11:01:33 PM
Duration	0:04:46
Run Id	diabetes-training_1586228492_66fa0c88
Arguments	N/A
Accuracy	0.7891111111111111
AUC	0.8568509052814499



- Register the environment in the work space

```
# Register the environment
diabetes_env.register(workspace=ws)
```

```
{
    "name": "diabetes-experiment-env",
    "version": "1",
    "environmentVariables": {
        "EXAMPLE_ENV_VAR": "EXAMPLE_VALUE"
    },
    "python": {
        "userManagedDependencies": false,
        "interpreterPath": "python",
        "condaDependenciesFile": null,
        "baseCondaEnvironment": null,
        "condaDependencies": {
            "channels": [
                "anaconda",
                "conda-forge"
            ],
            "dependencies": [
                "python=3.6.2",
                {
                    "pip": [
```

Run an Experiment on a Remote Compute Target¶

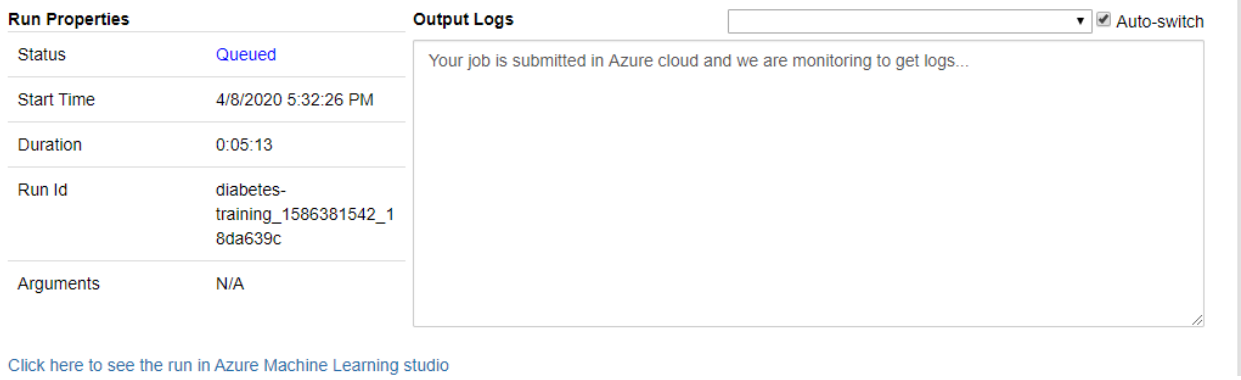
- Utilizing compute resources to run the experiment since the local computer might not be able to handle large amount of data

[8]:
```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

cluster_name = "aml-cluster"

try:
    # Check for existing compute target
    training_cluster = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    # If it doesn't already exist, create it
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_DS12_V2', max_nodes=4)
    training_cluster = ComputeTarget.create(ws, cluster_name, compute_config)

training_cluster.wait_for_completion(show_output=True)
```

```
Creating
Succeeded
AmlCompute wait for completion finished

Minimum number of nodes requested have been provisioned
```

- Running the experiment on the compute which was created. Since it is a free version of azure, the experiment was queued for a long time and was not scheduled to run. This step needs conda environment and assuming this takes longer time due to the free version.

```
experiment = Experiment(workspace = ws, name = 'diabetes-training')

# Run the experiment
run = experiment.submit(config=estimator)
# Show the run details while running
RunDetails(run).show()
run.wait_for_completion()
```



5. Creating an Azure Machine Learning Pipeline

- Connecting to the work space

```
n [1]: import azureml.core
       from azureml.core import Workspace

       # Load the workspace from the saved config file
       ws = Workspace.from_config()
       print('Ready to use Azure ML {} to work with {}'.format(azureml.core.VERSION, ws.name))

       Ready to use Azure ML 1.2.0 to work with greeshma1
```

- Registering the data base

```
    #Create a tabular dataset from the path on the datastore (thi
    tab_data_set = Dataset.Tabular.from_delimited_files(path=(def

    # Register the tabular dataset
    try:
        tab_data_set = tab_data_set.register(workspace=ws,
                                name='diabetes dataset',
                                description='diabetes data',
                                tags = {'format':'CSV'},
                                create_new_version=True)
        print('Dataset registered.')
    except Exception as ex:
        print(ex)
else:
    print('Dataset already registered.')
```

```
Dataset already registered.
```

- Creating a folder to contain the scripts for each step

[3]:
```
import os
# Create a folder for the pipeline step files
experiment_folder = 'diabetes_pipeline'
os.makedirs(experiment_folder, exist_ok=True)

print(experiment_folder)
```

```
diabetes_pipeline
```

```
auc = roc_auc_score(y_test,y_scores[:,1])
print('AUC: ' + str(auc))
run.log('AUC', np.float(auc))

# Save the trained model
os.makedirs(output_folder, exist_ok=True)
output_path = output_folder + "/model.pkl"
joblib.dump(value=model, filename=output_path)

run.complete()
```

```
Writing diabetes_pipeline/train_diabetes.py
```

```
# load the model
print("Loading model from " + model_folder)
model_file = model_folder + "/model.pkl"
model = joblib.load(model_file)

Model.register(workspace=run.experiment.workspace,
               model_path = model_file,
               model_name = 'diabetes_model',
               tags={'Training context':'Pipeline'})

run.complete()
```

```
Writing diabetes_pipeline/register_diabetes.py
```

Prepare a Compute Environment for the Pipeline Steps

- Creating a compute target

```
    compute_config = AmlCompute.provisioning_configuration(vm
                                                            ma
                                                            id
    pipeline_cluster = ComputeTarget.create(ws, cluster_name,

pipeline_cluster.wait_for_completion(show_output=True)
```

```
Found existing cluster, use it.
Succeeded
AmlCompute wait for completion finished

Minimum number of nodes requested have been provisioned
```

- Compute will require a Python environment with the necessary package dependencies installed

```
# Add the dependencies to the environment
diabetes_env.python.conda_dependencies = diabetes_packages

# Register the environment (just in case you want to use it again)
diabetes_env.register(workspace=ws)
registered_env = Environment.get(ws, 'diabetes-pipeline-env')

# Create a new runconfig object for the pipeline
pipeline_run_config = RunConfiguration()

# Use the compute you created above.
pipeline_run_config.target = pipeline_cluster

# Assign the environment to the run configuration
pipeline_run_config.environment = registered_env

print ("Run configuration created.")
```

```
Run configuration created.
```

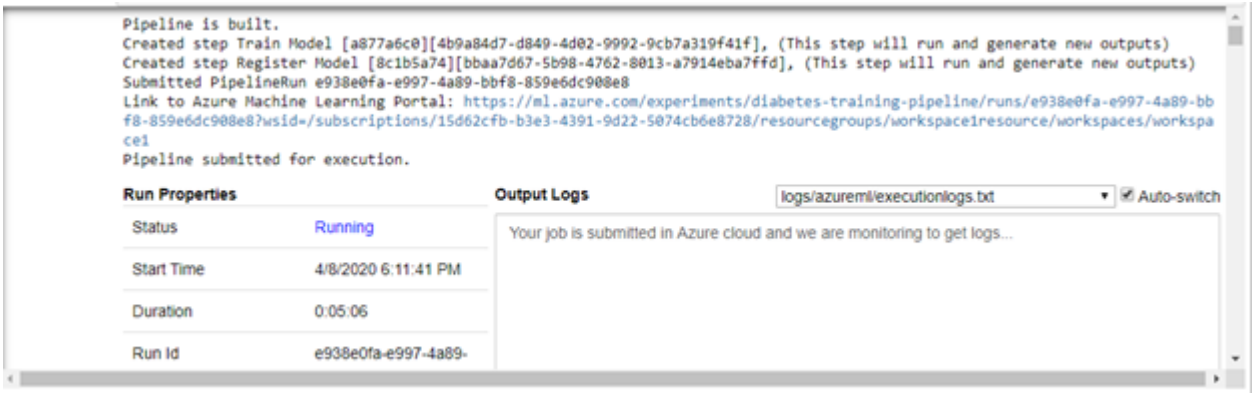
Create and Run a Pipeline

- Defining pipeline steps

```
# Create a PipelineData (temporary Data Reference) for the model folder
model_folder = PipelineData("model_folder", datastore=ws.get_default_datastore())

estimator = Estimator(source_directory=experiment_folder,
                      compute_target = pipeline_cluster,
                      environment_definition=pipeline_run_config.environment,
                      entry_script='train_diabetes.py')

# Step 1, run the estimator to train the model
train_step = EstimatorStep(name = "Train Model",
                           estimator=estimator,
                           estimator_entry_script_arguments=['--output_folder', model_folder],
                           inputs=[diabetes_ds.as_named_input('diabetes_train')],
                           outputs=[model_folder],
                           compute_target = pipeline_cluster,
                           allow_reuse = True)

# Step 2, run the model registration script
register_step = PythonScriptStep(name = "Register Model",
                                 source_directory = experiment_folder,
                                 script_name = "register_diabetes.py",
                                 arguments = ['--model_folder', model_folder],
                                 inputs=[model_folder],
                                 compute_target = pipeline_cluster,
                                 runconfig = pipeline_run_config,
                                 allow_reuse = True)

print("Pipeline steps defined")
```

```
Pipeline steps defined
```

- Running the pipeline. I left this to run for almost two days but I believe this will not be completed since it is a free azure account. I am assuming these jobs would be set priority and since it is a free account, it was not fully finished.



6. Creating a Real-Time Inferencing Service

- Connecting to work space

In [2]:
```
import azureml.core
from azureml.core import Workspace

# Load the workspace from the saved config file
ws = Workspace.from_config()
print('Ready to use Azure ML {} to work with {}'.format(azureml.core.VERSION, ws.name))
```

```
Ready to use Azure ML 1.2.0 to work with greeshma1
```

- Training the model to deploy

```
print('AUC: ' + str(auc))
run.log('AUC', np.float(auc))

# Save the trained model
model_file = 'diabetes_model.pkl'
joblib.dump(value=model, filename=model_file)
run.upload_file(name = 'outputs/' + model_file, path_or_stream = './' + model_file)

# Complete the run
run.complete()

# Register the model
run.register_model(model_path='outputs/diabetes_model.pkl', model_name='diabetes_model',
                   tags={'Training context':'Inline Training'},
                   properties={'AUC': run.get_metrics()['AUC'], 'Accuracy': run.get_metrics()['Accuracy']})

print('Model trained and registered.')
```

```
Starting experiment: diabetes-training
Loading Data...
Training a decision tree model
Accuracy: 0.8936666666666667
AUC: 0.8808423304642021
Model trained and registered.
```

Deploy a Model as a Web Service

- Determining what models you have registered in the workspace

```
for model in Model.list(ws):
    print(model.name, 'version:', model.version)
    for tag_name in model.tags:
        tag = model.tags[tag_name]
        print ('\t',tag_name, ':', tag)
    for prop_name in model.properties:
        prop = model.properties[prop_name]
        print ('\t',prop_name, ':', prop)
    print('\n')
```

```
diabetes_model version: 3
         Training context : Inline Training
         AUC : 0.8808423304642021
         Accuracy : 0.8936666666666667


diabetes_model version: 2
         Training context : Parameterized SKLearn Estimator
         AUC : 0.8483904671874223
         Accuracy : 0.7736666666666666


diabetes_model version: 1
         Training context : Estimator
         AUC : 0.8484929598487486
         Accuracy : 0.774
```

- Getting the model which has to be deployed

```
model = ws.models['diabetes_model']
print(model.name, 'version', model.version)
```

```
diabetes_model version 3
```

- Creating folder for code and configuration files

```
import os

folder_name = 'diabetes_service'

# Create a folder for the web service files
experiment_folder = './' + folder_name
os.makedirs(folder_name, exist_ok=True)

print(folder_name, 'folder created.')
```

```
diabetes_service folder created.
```

- The web service to deploy the model will need some Python code to load the input data, get the model from the workspace, and generate and return predictions. Saving this code in an entry script that will be deployed to the web service

```
    model = joblib.load(model_path)

# Called when a request is received
def run(raw_data):
    # Get the input data as a numpy array
    data = np.array(json.loads(raw_data)['data'])
    # Get a prediction from the model
    predictions = model.predict(data)
    # Get the corresponding classname for each prediction (0 or 1)
    classnames = ['not-diabetic', 'diabetic']
    predicted_classes = []
    for prediction in predictions:
        predicted_classes.append(classnames[prediction])
    # Return the predictions as JSON
    return json.dumps(predicted_classes)
```

```
Writing diabetes_service/score_diabetes.py
```

- Web service will be hosted in a container, and the container will need to install any required Python dependencies when it gets initialized. In this case, scoring code requires scikit-learn, so creating a .yml file that tells the container host to install this into the environment

```
env_file = folder_name + "/diabetes_env.yml"
with open(env_file,"w") as f:
    f.write(myenv.serialize_to_string())
print("Saved dependency info in", env_file)

# Print the .yml file
with open(env_file,"r") as f:
    print(f.read())
```

```
Saved dependency info in diabetes_service/diabetes_env.yml
# Conda environment specification. The dependencies defined in this file will
# be automatically provisioned for runs with userManagedDependencies=False.

# Details about the Conda environment file format:
# https://conda.io/docs/user-guide/tasks/manage-environments.html#create-env-file-manually

name: project_environment
dependencies:
  # The python interpreter version.
  # Currently Azure ML only supports 3.5.2 and later.
- python=3.6.2

- pip:
    # Required packages for AzureML execution, history, and data preparation.
  - azureml-defaults

- scikit-learn
channels:
```

- Deploying the model

```
]: from azureml.core.webservice import AciWebservice
   from azureml.core.model import InferenceConfig

   # Configure the scoring environment
   inference_config = InferenceConfig(runtime= "python",
                                      source_directory = folder_name,
                                      entry_script="score_diabetes.py",
                                      conda_file="diabetes_env.yml")

   deployment_config = AciWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)

   service_name = "diabetes-service"

   service = Model.deploy(ws, service_name, [model], inference_config, deployment_config)

   service.wait_for_deployment(True)
   print(service.state)
```

```
Running.....................................................................................
Succeeded
ACI service creation operation finished, operation "Succeeded"
Healthy
```

```
print(service.state)
print(service.get_logs())

# If you need to make a change and redeploy, you may need to delete
#service.delete()
```

```
Healthy
2020-04-07T04:17:11,760622098+00:00 - rsyslog/run
2020-04-07T04:17:11,766928615+00:00 - gunicorn/run
2020-04-07T04:17:11,771446726+00:00 - nginx/run
/usr/sbin/nginx: /azureml-envs/azureml_4b824bcb98517d791c41923f24d6
ion available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_4b824bcb98517d791c41923f24d6
ion available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_4b824bcb98517d791c41923f24d6
available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_4b824bcb98517d791c41923f24d6
available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_4b824bcb98517d791c41923f24d6
available (required by /usr/sbin/nginx)
2020-04-07T04:17:11,781348352+00:00 - iot-server/run
EdgeHubConnectionString and IOTEDGE_IOTHUBHOSTNAME are not set. Exi
2020-04-07T04:17:12,012840151+00:00 - iot-server/finish 1 0
```

```
: for webservice_name in ws.webservices:
      print(webservice_name)
```

```
diabetes-service
```

- The service deployed can be consumed it from a client application

```
import json

x_new = [[2,180,74,24,21,23.9091702,1.488172308,22]]
print ('Patient: {}'.format(x_new[0]))

# Convert the array to a serializable list in a JSON document
input_json = json.dumps({"data": x_new})

# Call the web service, passing the input data (the web service
predictions = service.run(input_data = input_json)

# Get the predicted class - it'll be the first (and only) one.
predicted_classes = json.loads(predictions)
print(predicted_classes[0])
```

```
Patient: [2, 180, 74, 24, 21, 23.9091702, 1.488172308, 22]
diabetic
```

- Sending multiple patient observations to the service, and get back a prediction for each one

```
import json

# This time our input is an array of two feature arrays
x_new = [[2,180,74,24,21,23.9091702,1.488172308,22],
         [0,148,58,11,179,39.19207553,0.160829008,45]]

# Convert the array or arrays to a serializable list in a JSON document
input_json = json.dumps({"data": x_new})

# Call the web service, passing the input data
predictions = service.run(input_data = input_json)

# Get the predicted classes.
predicted_classes = json.loads(predictions)

for i in range(len(x_new)):
    print ("Patient {}".format(x_new[i]), predicted_classes[i] )
```

```
Patient [2, 180, 74, 24, 21, 23.9091702, 1.488172308, 22] diabetic
Patient [0, 148, 58, 11, 179, 39.19207553, 0.160829008, 45] not-diabetic
```

- Determining the URL to which these applications must submit their requests

```
endpoint = service.scoring_uri
print(endpoint)
```

```
http://15f8aded-67fc-4c6a-9ce9-917ade620918.eastus2.azurecontainer.io/score
```

- An application can simply make an HTTP request, sending the patient data in JSON (or binary) format, and receive back the predicted class(es)

```
import requests
import json

x_new = [[2,180,74,24,21,23.9091702,1.488172308,22],
         [0,148,58,11,179,39.19207553,0.160829008,45]]

# Convert the array to a serializable list in a JSON document
input_json = json.dumps({"data": x_new})

# Set the content type
headers = { 'Content-Type':'application/json' }

predictions = requests.post(endpoint, input_json, headers = headers)
predicted_classes = json.loads(predictions.json())

for i in range(len(x_new)):
    print ("Patient {}".format(x_new[i]), predicted_classes[i] )
```

```
Patient [2, 180, 74, 24, 21, 23.9091702, 1.488172308, 22] diabetic
Patient [0, 148, 58, 11, 179, 39.19207553, 0.160829008, 45] not-diabetic
```

7. Using Automated Machine Learning

- Connecting to work space

[1]:
```
import azureml.core
from azureml.core import Workspace

# Load the workspace from the saved config file
ws = Workspace.from_config()
print('Ready to use Azure ML {} to work with {}'.format(azureml.core.VERSION, ws.name))
```

```
Ready to use Azure ML 1.2.0 to work with greeshma1
```

- Running configuration that includes the required packages for the experiment environment, configuration settings that specifies how many combinations to try, which metric to use when evaluating models

```
[2]: import pandas as pd
     from azureml.train.automl import AutoMLConfig

     # Load the data
     train_data = pd.read_csv('data/diabetes.csv')

     automl_config = AutoMLConfig(name='Automated ML Experiment',
                                  task='classification',
                                  compute_target='local',
                                  training_data = train_data,
                                  n_cross_validations = 2,
                                  label_column_name='Diabetic',
                                  iterations=6,
                                  primary_metric = 'AUC_weighted',
                                  max_concurrent_iterations=3,
                                  featurization='auto'
                                  )

     print("Ready for Auto ML run.")
```

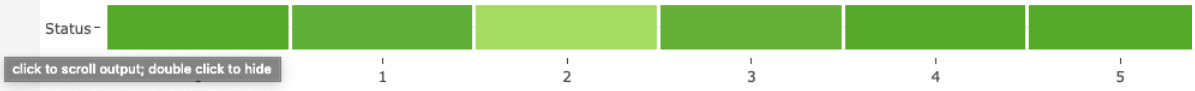
```
Ready for Auto ML run.
```

- Running the automated machine learning experiment

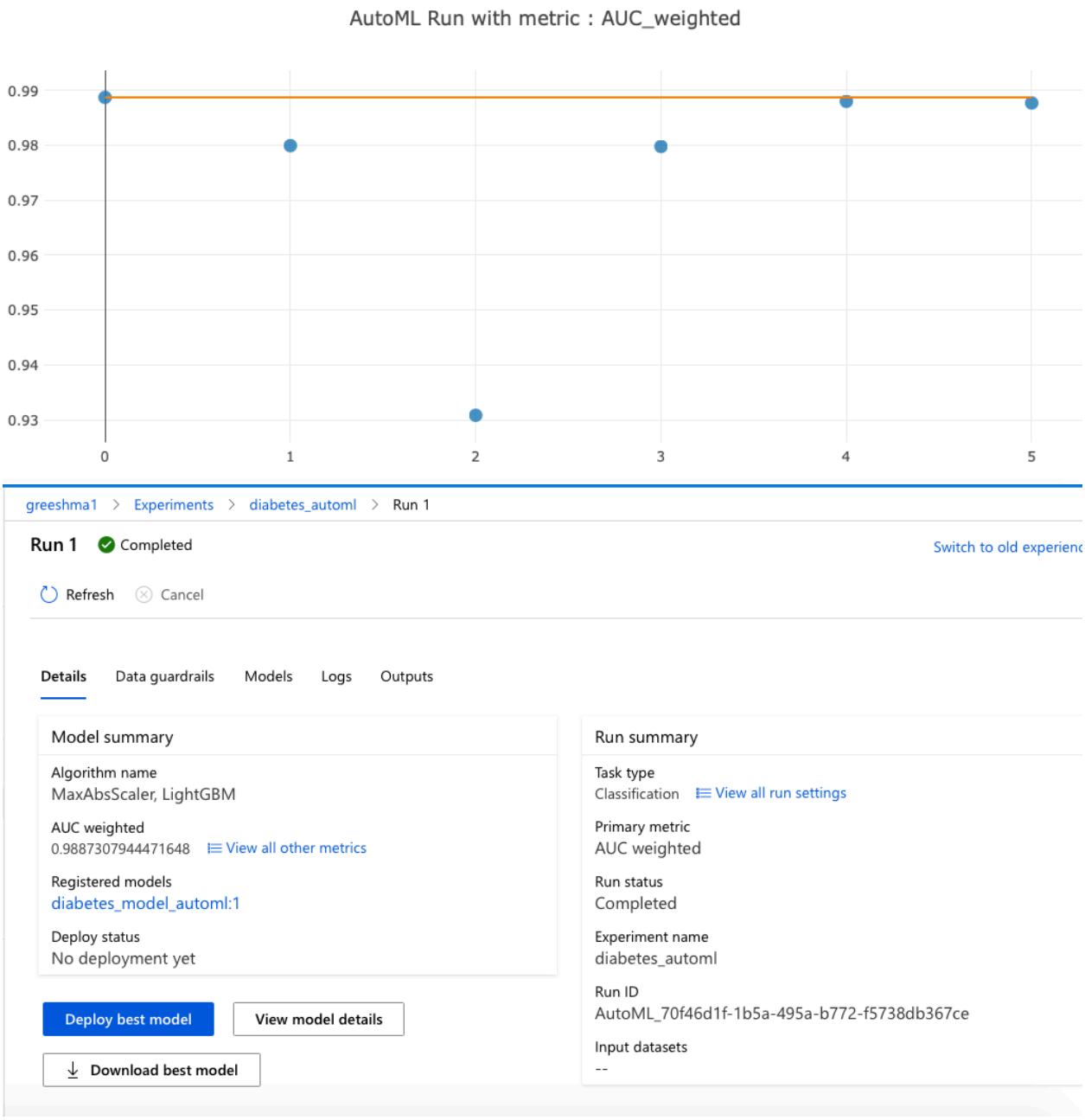
```
In [3]: from azureml.core.experiment import Experiment
        from azureml.widgets import RunDetails

        print('Submitting Auto ML experiment...')
        automl_experiment = Experiment(ws, 'diabetes_automl')
        automl_run = automl_experiment.submit(automl_config)
        RunDetails(automl_run).show()
        automl_run.wait_for_completion()
```

```
Submitting Auto ML experiment...
```

AutoML_70f46d1f-1b5a-495a-b772-f5738db367ce:
Status: Completed



Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started
0	MaxAbsScaler, LightGBM	0.98873079	0.98873079	Completed	0:00:22	Apr 7, 2020 12:37 AM
4	VotingEnsemble	0.98801063	0.98873079	Completed	0:00:13	Apr 7, 2020 12:38 AM
5	StackEnsemble	0.9877102	0.98873079	Completed	0:00:12	Apr 7, 2020 12:39 AM
1	MaxAbsScaler, LightGBM	0.97994864	0.98873079	Completed	0:00:21	Apr 7, 2020 12:37 AM



- Determining the best model

```
best_run, fitted_model = automl_run.get_output()
print(best_run)
print(fitted_model)
best_run_metrics = best_run.get_metrics()
for metric_name in best_run_metrics:
    metric = best_run_metrics[metric_name]
    print(metric_name, metric)
```

```
Run(Experiment: diabetes_automl,
Id: AutoML_70f46d1f-1b5a-495a-b772-f5738db367ce_0,
Type: None,
Status: Completed)
Pipeline(memory=None,
     steps=[('datatransformer', DataTransformer(enable_dnn=None, enable_feature_sweeping=None,
        feature_sweeping_config=None, feature_sweeping_timeout=None,
        featurization_config=None, force_text_dnn=None,
        is_cross_validation=None, is_onnx_compatible=None, logger=None,
        obser...    silent=True, subsample=1.0, subsample_for_bin=200000,
          subsample_freq=0, verbose=-10))])
norm_macro_recall 0.8794903988906442
average_precision_score_weighted 0.9893133642855426
average_precision_score_macro 0.9868730917793174
precision_score_macro 0.9432823058330543
```

- Viewing the steps in a model

```
for step in fitted_model.named_steps:
    print(step)
```

```
datatransformer
MaxAbsScaler
LightGBMClassifier
```

- Registering the best performing model

```
    for tag_name in model.tags:
        tag = model.tags[tag_name]
        print ('\t',tag_name, ':', tag)
    for prop_name in model.properties:
        prop = model.properties[prop_name]
        print ('\t',prop_name, ':', prop)
    print('\n')
```

```
diabetes_model_automl version: 1
        Training context : Auto ML
        AUC : 0.9887307944471648
        Accuracy : 0.9480999999999999


diabetes_model version: 3
        Training context : Inline Training
        AUC : 0.8808423304642021
        Accuracy : 0.8936666666666667


diabetes_model version: 2
        Training context : Parameterized SKLearn Estimator
        AUC : 0.8483904671874223
        Accuracy : 0.7736666666666666


diabetes_model version: 1
        Training context : Estimator
        AUC : 0.8484929598487486
        Accuracy : 0.774
```

- Many experiments were run as a part of implementing machine learning models in different ways. Out of all the different ways, Auto ML was giving the highest accuracy of 94% for the given image data set.