QUESTION 130
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
You are using Azure Machine Learning to run an experiment that trains a classification model.
You want to use Hyperdrive to find parameters that optimize the AUC metric for the model. You configure a HyperDriveConfig for the experiment by running the following code:

```
hyperdrive = HyperDriveConfig(estimator=your_estimator,
 hyperparameter_sampling=your_params,
 policy=policy,
 primary_metric_name='AUC',
 primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
 max_total_runs=6,
 max_concurrent_runs=4)
```

You plan to use this configuration to run a script that trains a random forest model and then tests it with validation data. The label values for the validation data are stored in a variable named y_test variable, and the predicted probabilities from the model are stored in a variable named y_predicted.
You need to add logging to the script to allow Hyperdrive to optimize hyperparameters for the AUC metric.
Solution: Run the following code:

```
from sklearn.metrics import roc_auc_score
import logging
# code to train model omitted
auc = roc_auc_score(y_test, y_predicted)
logging.info("AUC: " + str(auc))
```

Does the solution meet the goal?

A. Yes
B. No

Answer: A
Explanation:
Python printing/logging example:
logging.info(message)
Destination: Driver logs, Azure Machine Learning designer
Reference:

https://docs.microsoft.com/en-us/azure/machine-learning/how-to-debug-pipelines

QUESTION 131
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
You are using Azure Machine Learning to run an experiment that trains a classification model.
You want to use Hyperdrive to find parameters that optimize the AUC metric for the model. You configure a HyperDriveConfig for the experiment by running the following code:

```
hyperdrive = HyperDriveConfig(estimator=your_estimator,
 hyperparameter_sampling=your_params,
 policy=policy,
 primary_metric_name='AUC',
 primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
 max_total_runs=6,
 max_concurrent_runs=4)
```

You plan to use this configuration to run a script that trains a random forest model and then tests it with validation data. The label values for the validation data are stored in a variable named y_test variable, and the predicted probabilities from the model are stored in a variable named y_predicted.
You need to add logging to the script to allow Hyperdrive to optimize hyperparameters for the AUC metric.
Solution: Run the following code:

```
import json, os
from sklearn.metrics import roc_auc_score
# code to train model omitted
auc = roc_auc_score(y_test, y_predicted)
os.makedirs("outputs", exist_ok = True)
with open("outputs/AUC.txt", "w") as file_cur:
    file_cur.write(auc)
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B
Explanation:
Use a solution with logging.info(message) instead.
Note: Python printing/logging example:
logging.info(message)
Destination: Driver logs, Azure Machine Learning designer
Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-debug-pipelines

QUESTION 132
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
You are using Azure Machine Learning to run an experiment that trains a classification model.
You want to use Hyperdrive to find parameters that optimize the AUC metric for the model. You configure a HyperDriveConfig for the experiment by running the following code:

```
hyperdrive = HyperDriveConfig(estimator=your_estimator,
 hyperparameter_sampling=your_params,
 policy=policy,
 primary_metric_name='AUC',
 primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
 max_total_runs=6,
 max_concurrent_runs=4)
```

You plan to use this configuration to run a script that trains a random forest model and then tests it with validation data. The label values for the validation data are stored in a variable named y_test variable, and the predicted probabilities from the model are stored in a variable named y_predicted.
You need to add logging to the script to allow Hyperdrive to optimize hyperparameters for the AUC metric.
Solution: Run the following code:

```
import numpy as np
from sklearn.metrics import roc_auc_score
# code to train model omitted
auc = roc_auc_score(y_test, y_predicted)
print(np.float(auc))
```

Does the solution meet the goal?

A. Yes
B. No

Answer: B
Explanation:
Use a solution with logging.info(message) instead.
Note: Python printing/logging example:
logging.info(message)
Destination: Driver logs, Azure Machine Learning designer
Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-debug-pipelines

QUESTION 133
You plan to provision an Azure Machine Learning Basic edition workspace for a data science project. You need to identify the tasks you will be able to perform in the workspace.
Which three tasks will you be able to perform? Each correct answer presents a complete solution.
NOTE: Each correct selection is worth one point.

A. Create a Compute Instance and use it to run code in Jupyter notebooks.
B. Create an Azure Kubernetes Service (AKS) inference cluster.
C. Use the designer to train a model by dragging and dropping pre-defined modules.
D. Create a tabular dataset that supports versioning.
E. Use the Automated Machine Learning user interface to train a model.

Answer: ABD
Explanation:
Incorrect Answers:
C, E: The UI is included the Enterprise edition only.
Reference:
https://azure.microsoft.com/en-us/pricing/details/machine-learning/

QUESTION 134
A set of CSV files contains sales records. All the CSV files have the same data schema.
Each CSV file contains the sales record for a particular month and has the filename sales.csv. Each file in stored in a

folder that indicates the month and year when the data was recorded. The folders are in an Azure blob container for which a datastore has been defined in an Azure Machine Learning workspace. The folders are organized in a parent folder named sales to create the following hierarchical structure:

```
/sales
  /01-2019
    /sales.csv
  /02-2019
    /sales.csv
  /03-2019
    /sales.csv
  ...
```

At the end of each month, a new folder with that month's sales file is added to the **sales** folder.
You plan to use the sales data to train a machine learning model based on the following requirements:

- You must define a dataset that loads all of the sales data to date into a structure that can be easily converted to a dataframe.
- You must be able to create experiments that use only data that was created before a specific previous month, ignoring any data that was added after that month.
- You must register the minimum number of datasets possible.

You need to register the sales data as a dataset in Azure Machine Learning service workspace.
What should you do?

A. Create a tabular dataset that references the datastore and explicitly specifies each 'sales/mm-yyyy/ sales.csv' file every month.
Register the dataset with the name sales_dataset each month, replacing the existing dataset and specifying a tag named month indicating the month and year it was registered.
Use this dataset for all experiments.

B. Create a tabular dataset that references the datastore and specifies the path 'sales/*/sales.csv', register the dataset with the name sales_dataset and a tag named month indicating the month and year it was registered, and use this dataset for all experiments.

C. Create a new tabular dataset that references the datastore and explicitly specifies each 'sales/mm- /sales.csv' file every month.
Register the dataset with the name sales_dataset_MM-YYYY each yyyy month with appropriate MM and YYYY values for the month and year.
Use the appropriate month-specific dataset for experiments.

D. Create a tabular dataset that references the datastore and explicitly specifies each 'sales/mm-yyyy/ sales.csv' file.
Register the dataset with the name sales_dataset each month as a new version and with a tag named month indicating the month and year it was registered.
Use this dataset for all experiments, identifying the version to be used based on the month tag as necessary.

Answer: B
Explanation:
Specify the path.
Example:
The following code gets the workspace existing workspace and the desired datastore by name. And then passes the datastore and file locations to the path parameter to create a new TabularDataset, weather_ds.

```
from azureml.core import Workspace, Datastore, Dataset
datastore_name = 'your datastore name'
# get existing workspace
workspace = Workspace.from_config()
# retrieve an existing datastore in the workspace by name datastore = Datastore.get(workspace, datastore_name)
# create a TabularDataset from 3 file paths in datastore datastore_paths = [(datastore, 'weather/2018/11.csv'),
(datastore, 'weather/2018/12.csv'),
```

(datastore, 'weather/2019/*.csv')]
weather_ds = Dataset.Tabular.from_delimited_files(path=datastore_paths)

QUESTION 135
You use the following code to run a script as an experiment in Azure Machine Learning:

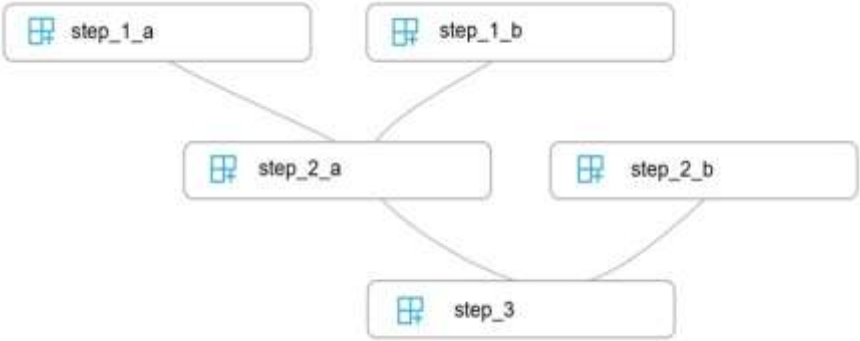
```
from azureml.core import Workspace, Experiment, Run
from azureml.core import RunConfig, ScriptRunConfig
ws = Workspace.from_config()
run_config = RunConfiguration()
run_config.target='local'
script_config = ScriptRunConfig(source_directory='./script', script='experiment.py', run_config=run_config)
experiment = Experiment(workspace=ws, name='script experiment')
run = experiment.submit(config=script_config)
run.wait_for_completion()
```

You must identify the output files that are generated by the experiment run.
You need to add code to retrieve the output file names.
Which code segment should you add to the script?

A. files = run.get_properties()
B. files= run.get_file_names()
C. files = run.get_details_with_logs()
D. files = run.get_metrics()
E. files = run.get_details()

Answer: B
Explanation:
You can list all of the files that are associated with this run record by called run.get_file_names() Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-track-experiments

QUESTION 136
You write five Python scripts that must be processed in the order specified in Exhibit A – which allows the same modules to run in parallel, but will wait for modules with dependencies.
You must create an Azure Machine Learning pipeline using the Python SDK, because you want to script to create the pipeline to be tracked in your version control system. You have created five PythonScriptSteps and have named the variables to match the module names.



You need to create the pipeline shown. Assume all relevant imports have been done.
Which Python code segment should you use?

A. `p = Pipeline(ws, steps=[[[[step_1_a, step_1_b], step_2_a], step_2_b], step_3])`

B.
```
pipeline_steps = {
    "Pipeline": {
        "run": step_3,
        "run_after": {[
            {"run": step_2_a,
                "run_after":
                    [{"run": step_1_a},
                     {"run": step_1_b}]
                },
            {"run": step_2_b}]
        }
    }
}
p = Pipeline(ws, steps=pipeline_steps)
```

C.
```
step_2_a.run_after(step_1_b)
step_2_a.run_after(step_1_a)
step_3.run_after(step_2_b)
step_3.run_after(step_2_a)
p = Pipeline(ws, steps=[step_3])
```

D.
```
p = Pipeline(ws, steps=[step_1_a, step_1_b, step_2_a, step_2_b, step_3])
```

Answer: A
Explanation:
The steps parameter is an array of steps. To build pipelines that have multiple steps, place the steps in order in this array.
Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-parallel-run-step

QUESTION 137
You create a datastore named **training_data** that references a blob container in an Azure Storage account. The blob container contains a folder named **csv_files** in which multiple comma-separated values (CSV) files are stored.
You have a script named train.py in a local folder named ./script that you plan to run as an experiment using an estimator. The script includes the following code to read data from the csv_files folder:

```
import os
import argparse
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from azureml.core import Run

run = Run.get_context()
parser = argparse.ArgumentParser()
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data reference')
args = parser.parse_args()

data_folder = args.data_folder
csv_files = os.listdir(data_folder)
training_data = pd.concat((pd.read_csv(os.path.join(data_folder,csv_file)) for csv_file in csv_files))

# Code goes on to split the training data and train a logistic regression model
```

You have the following script.

```
from azureml.core import Workspace, Datastore, Experiment
from azureml.train.sklearn import SKLearn

ws = Workspace.from_config()
exp = Experiment(workspace=ws, name='csv_training')
ds = Datastore.get(ws, datastore_name='training_data')
data_ref = ds.path('csv_files')

# Code to define estimator goes here

run = exp.submit(config=estimator)
run.wait_for_completion(show_output=True)
```

You need to configure the estimator for the experiment so that the script can read the data from a data reference named data_ref that references the csv_files folder in the training_data datastore.
Which code should you use to configure the estimator?

A.
```
estimator = SKLearn(source_directory='./script',
 inputs=[data_ref.as_named_input('data-folder').to_pandas_dataframe()],
 compute_target='local',
 entry_script='train.py')
```

B.
```
script_params = {
    '--data-folder': data_ref.as_mount()
}
estimator = SKLearn(source_directory='./script',
 script_params=script_params,
 compute_target='local',
 entry_script='train.py'
```

C.
```
estimator = SKLearn(source_directory='./script',
 inputs=[data_ref.as_named_input('data-folder').as_mount()],
 compute_target='local',
 entry_script='train.py')
```

D.
```
script_params = {
    '--data-folder': data_ref.as_download(path_on_compute='csv_files')
}
estimator = SKLearn(source_directory='./script',
 script_params=script_params,
 compute_target='local',
 entry_script='train.py'
```

E.
```
estimator = SKLearn(source_directory='./script',
 inputs=[data_ref.as_named_input('data-folder').as_download(path_on_compute='csv_files')],
 compute_target='local',
 entry_script='train.py')
```

Answer: B
Explanation:
Besides passing the dataset through the inputs parameter in the estimator, you can also pass the dataset through script_params and get the data path (mounting point) in your training script via arguments. This way, you can keep your training script independent of azureml-sdk. In other words, you will be able use the same training script for local debugging and remote training on any cloud platform.
Example:
from azureml.train.sklearn import SKLearn
script_params = {
# mount the dataset on the remote compute and pass the mounted path as an argument to the training script
'--data-folder': mnist_ds.as_named_input('mnist').as_mount(), '--regularization': 0.5
}
est = SKLearn(source_directory=script_folder,
script_params=script_params,
compute_target=compute_target,

environment_definition=env,
entry_script='train_mnist.py')
# Run the experiment
run = experiment.submit(est)
run.wait_for_completion(show_output=True)
Incorrect Answers:
A: Pandas DataFrame not used.
Reference:
https://docs.microsoft.com/es-es/azure/machine-learning/how-to-train-with-datasets

QUESTION 138
You create a deep learning model for image recognition on Azure Machine Learning service using GPU- based training.
You must deploy the model to a context that allows for real-time GPU-based inferencing.
You need to configure compute resources for model inferencing.
Which compute type should you use?

A. Azure Container Instance
B. Azure Kubernetes Service
C. Field Programmable Gate Array
D. Machine Learning Compute

Answer: B
Explanation:
You can use Azure Machine Learning to deploy a GPU-enabled model as a web service. Deploying a model on Azure Kubernetes Service (AKS) is one option. The AKS cluster provides a GPU resource that is used by the model for inference.
Inference, or model scoring, is the phase where the deployed model is used to make predictions. Using GPUs instead of CPUs offers performance advantages on highly parallelizable computation.
Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-inferencing-gpus

QUESTION 139
You create a batch inference pipeline by using the Azure ML SDK. You run the pipeline by using the following code:

```
from azureml.pipeline.core import Pipeline
from azureml.core.experiment import Experiment
pipeline = Pipeline(workspace=ws, steps=[parallelrun_step])
pipeline_run = Experiment(ws, 'batch_pipeline').submit(pipeline)
```

You need to monitor the progress of the pipeline execution.
What are two possible ways to achieve this goal? Each correct answer presents a complete solution.
NOTE: Each correct selection is worth one point.

A. Run the following code in a notebook:

```
from azureml.contrib.interpret.explanation.explanation_client import ExplanationClient
client = ExplanationClient.from_run(pipeline_run)
explanation = client.download_model_explanation()
explanation = client.download_model_explanation(top_k=4)
global_importance_values = explanation.get_ranked_global_values()
global_importance_names = explanation.get_ranked_global_names()
print('global importance values: {}'.format(global_importance_values))
print('global importance names: {}'.format(global_importance_names))
```

B. Use the Inference Clusters tab in Machine Learning Studio.
C. Use the Activity log in the Azure portal for the Machine Learning workspace.
D. Run the following code in a notebook:

```
from azureml.widgets import RunDetails
RunDetails(pipeline_run).show()
```

E. Run the following code and monitor the console output from the PipelineRun object:

```
pipeline_run.wait_for_completion(show_output=True)
```

Answer: DE

Explanation:
A batch inference job can take a long time to finish. This example monitors progress by using a Jupyter widget. You can also manage the job's progress by using:
Azure Machine Learning Studio.
Console output from the PipelineRun object.
from azureml.widgets import RunDetails
RunDetails(pipeline_run).show()
pipeline_run.wait_for_completion(show_output=True)
Reference:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-parallel-run-step#monitor-the-parallel-run-job

QUESTION 140
You train and register a model in your Azure Machine Learning workspace.
You must publish a pipeline that enables client applications to use the model for batch inferencing. You must use a pipeline with a single ParallelRunStep step that runs a Python inferencing script to get predictions from the input data.
You need to create the inferencing script for the ParallelRunStep pipeline step.
Which two functions should you include? Each correct answer presents part of the solution.
NOTE: Each correct selection is worth one point.

A. run(mini_batch)
B. main()
C. batch()
D. init()
E. score(mini_batch)

Answer: AD
Explanation:
https://github.com/Azure/MachineLearningNotebooks/tree/master/how-to-use-azureml/machine-learning-pipelines/parallel-run