# Azure data scientist associate

- ExamTopics sample questions page 2
- Udemy practice exams

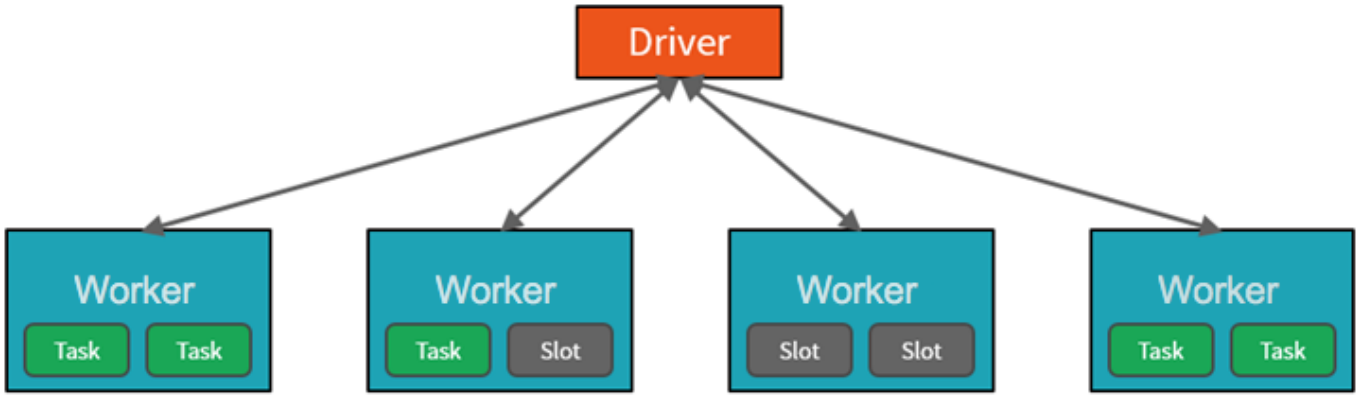


## 1. Azure Databricks

It's a big data and machine learning platform, fully-managed version of the open-source Apache Spark. *Azure Databricks is fully integrated with Azure Data Factory*.

Azure Data Factory is a ETL service for scale-out serverless data integration and transformation. Used for orchestration for production, for testing use container instances

1. Launch a workspace
2. Launch a cluster
    - Always has one Driver
    - Choose how many executors (Workers)
    - Choose types and sizes of the VMs
    - Choose Databricks runtime (you only choose VM config, the rest is done by Databricks)
    - Choose between 2 types of clusters
        - Standard: can be used with Python, R, Scala, SQL
        - High-concurrency
3. Launch notebook
    - Attach to cluster, attached_cluster can be changed
    - Cluster can be restarted from the notebook
    - Spark UI can be opened
    - If attached to cluster, driver's logs can be seen
    - Format of the notebook is DBC
    - Notebooks are a collection of cells




## Levels of parallelization

- **Executor**, first level of parallelization: a **Java** virtual machine running on a node, typically, one instance per node
    - Each Executor has a number of Slots to which parallelized Tasks can be assigned to it by the Driver
- **Slot**, second level of parallelization: the number of which is determined by the number of cores and CPUs of each node

## Managing work

- Work is submitted to the cluster and split into Jobs (distributed across nodes)
- Each parallelized action is referred to as a Job
- The results of each Job (parallelized/distributed action) is returned to the Driver
- Depending on the work required, multiple Jobs will be required
- Each Job is broken down into Stages
- In between tasks, partitions may need to be re-organized and shared over the network

### UDFs

UDFs allow you to encapsulate your complex and custom data processing logic. As a result UDFs make custom logic reusable, easy to understand, maintain, and debug. But they are serialized and it's better to use built-in functions.

- To register the UDF f as my_function: `spark.udf.register("my_function", f)`
- To specify the return type, `my_udf = udf(f, "long")`

UDFs can operate several rows at a time, but can only return one rowset to the user. They can't use non-deterministic built-in functions and the function has to be serialized and sent out to executors.

### Frameworks

- *Rattle*: R analytical tool to get started with data analytics and machine learning
- *Weka*: Java tool for visual data mining and machine learning, first convert to ARFF format
- *Scikit-learn*: Python tool for machine learning
- *TensorFlow*: an open-source library for numerical computation and large-scale machine learning. It uses Python to provide a front-end API

### Code snippets

To infer data types and column names from JSON:

```
spark.read.option("inferSchema", "true").json(jsonFile)
df=spark.read.format("json").option("inferSchema”,"true").load(filePath)
```

To move data into the memory of the local executor: `.cache()`. cache(), persist(), cacheTable() are lazily evaluated and need to perform an action to work whereas SQL CACHE TABLE is an eager.

To create temporary views in DataFrames: `createOrReplaceTempView()`

### DBFS

Databricks File System (DBFS) is a distributed file system mounted into a Databricks workspace and available on Databricks clusters.

- Allows you to mount storage objects to access data without requiring credentials.
- Allows you to interact with object storage using directory and file semantics instead of storage URLs.
- Persists files to object storage, so you won’t lose data after you terminate a cluster.

To list files in DBFS within a notebook: `%fs ls /my-file-path`.

### Apache Zeppelin

Web-based notebook for data exploration, visualization, sharing and collaboration features to Spark. It runs on HDInsight Spark clusters. The notebook runs Spark jobs.

1. Create HDInsight Spark cluster to include Spark MLib library
2. Install Microsoft ML for Spark

3. Create and execute Zeppelin notebooks on cluster
4. When cluster is ready, export notebooks to local

If model needs to be trained from scratch with deep learning techniques, use Azure HDInsight with Spark MLib.

---

# 2. Azure Machine Learning

Subscription types:

- Azure ML Basic: supports notebooks, compute instances, AKS inference clusters, creating datasets.
- Azure ML Enterprise: supports UI, designers and AutoML

Can be managed by:

- Azure ML studio (general page of AML)
- Azure ML designer (no-code site)
- Azure ML SDK

To create a **workspace**, recommended to use a config file, which needs: workspace_name, resource_group and subscription_id.

## 2.1. Data

**Datastores** can be viewed and managed via: AML SDK, and AML designer. Not via CLI, PowerShell. Valid types of datastore: DBFS, Blob storage, SQL database, data lake stores. Not Azure Cosmos

Data can be uploaded to Azure Blob storage via Azure Storage-Explorer, AzCopy, Python, SQL Server Integration Services.

To register a **dataset**: AML designer or AML SDK. Paths can include wildcards (/files/*.csv) to create just one dataset

- Tabular: data is read from the dataset as a table. Used for structured data, Pandas dataframes. CSV is common but Parquet format has better performance

```
blob_ds = ws.get_default_datastore()
# opening data from 2 paths
csv_paths = [(blob_ds, 'data/current_data.csv'),
             (blob_ds, 'data/archive/*.csv')]
tab_ds = Dataset.Tabular.from_delimited_files(path=csv_paths)
tab_ds = tab_ds.register(workspace=ws, name='csv_table')
# to convert dataset to df
df = tab_ds.to_pandas_dataframe()
```

- File: a list of file paths. Used for unstructured data, or images

```
blob_ds = ws.get_default_datastore()
file_ds = Dataset.File.from_files(path=(blob_ds, 'data/images/*.jpg'))
file_ds = file_ds.register(workspace=ws, name='img_files')
# to create new version, add create_new_version=True to .register()
```

To get a dataset:

```
ds2 = Dataset.get_by_name(ws, 'img_files')
# to get specific version, add version=2 to .get_by_name()
```

When there are files for each month and you want to have only one dataset:

- Create a tabular dataset that references the datastore and explicitly specifies each 'sales/mm-yyyy/sales.csv' file
- Register 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

A dataset can be passed as a script argument, in that case you need to pass the unique ID for the dataset

```
env = Environment('my_env')
packages = CondaDependencies.create(conda_packages=['pip'],
pip_packages=['azureml-defaults', 'azureml-dataprep[pandas]'])
env.python.conda_dependencies = packages
script_config = ScriptRunConfig(source_directory='my_dir',
                                script='script.py',
                                arguments=['--ds', tab_ds],
# or as named input: tab_ds.as_named_input('my_dataset')
                                environment=env
                                )
```

To check data drift in a dataset, define an AlertConfiguration and set a drift_threshold value. If this is fulfilled, you can configure alerts to email.

To track changing data, create a data drift monitor that uses the traindata as a baseline and the new data as target. After creating it, you can backfill to immediately compare the baseline dataset to the target dataset over a time period

## 2.2. Compute

- **Microsoft Cognitive Toolkit (CNTK)**: open-source toolkit for commercial-grade distributed DL. It describes neural networks as a series of computational steps via a directed graph. CNTK allows the user to easily realize and combine popular model types
- **Power BI Desktop**: visual data exploration and interactive reporting tool. BI is a name given to a modern approach to business decision-making in which users are empowered to find, explore, and share insights from data across the enterprise

- To do data visualization in the cloud, use Github Codespaces
- To install **Docker** on Windows: Windows system should support hardware virtualization tech, and BIOS-enabled virtualization. 64-bit OS running Windows 7 or higher.

Types of compute:

- Compute instance: development workstations to work with data and models
- Compute cluster: to train models using the AML designer, scalable
- Inference cluster: to score new data, and to deploy models
- Attached compute: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.
- Container instance: can be used as compute target for testing or development. Use for low-scale CPU-based workloads that require less than 48 GB of RAM.
- Machine Learning compute: for AML Designer.

Virtual machines:

- **Data Science Virtual Machine (DSVM)**: customized VM image on Microsoft's Azure cloud built specifically for doing data science. GPUs are supported. Caffe2, Chainer, PostgreSQL are supported by DSVM. Caffe2 and PyTorch is supported by Data Science Virtual Machine for Linux.
- **Deep Learning Virtual Machine (DLVM)**: pre-configured environment for deep learning using GPU instances
    - DLVM is a template on top of DSVM image. In terms of the packages, GPU drivers etc. are all there in the DSVM image. In general, it is for convenience during creation where we only allow DLVM to be created on GPU VM instances on Azure.
    - Seems to be no reason to use DLVM. All questions with DLVM, the correct answer is DSVM

## 2.3. Azure ML designer, pipelines

No-code UI only available for AML enterprise subscription. allows creating of pipelines. To pass data between steps in a pipeline, use `PipelineData` object.

After creating and running a pipeline to train the model, you can create an inference pipeline that uses the model to predict. First the training pipeline must be converted into a real-time inference pipeline. This removes training modules and adds web service inputs and outputs to handle requests.

**Splitting tables**

- Split rows with randomized split parameter: dividing data into two, the randomized_split parameter lets you specify the percentage for each split
- Regular expression split: to divide the dataset by testing a single column for a value
- Relative expression split: to apply a condition to a number column

**Data processing methods**

Binning:

- **Entropy Minimum Description Length (MDL)** binning mode: used to normalize values to produce a) an output column into bins to predict a target column, b) normalize values to produce a feature column grouped into bins. requires that you select the target column and the columns to be gropued into bins.

It looks at the data and attemps to determine the number of bins to minimize entropy. It then returns the bin number associated with each row of your data in a column named quantized
    - If the Entropy MDL method cannot find a way to initially bin the data to make a good prediction, it assigns all data to a uniform bin
    - This method doesn't return the actual entropy scores
- **Quantiles normalization with QuantileIndex** binning mode: doesn't use the target column. Use the Quantile normalization option to determine how values are normalized prior to sorting into quantiles. Note that normalizing values transforms the values, but does not affect the final number of bins.
- **Group Data into Bins**: multiple options for binning data. You can customize how the bin edges are set and how values are apportioned into the bins.

Replacing data

- **Multiple Imputation by Chained Equations (MICE)**: replaces missing data. With a multiple imputation method, each variable with missing data is modeled conditionally using the other variables in the data before filling in the missing values. Dimensionality of the table is not changed. This method requires the application of predictors for each column.
- **Probabilistic PCA**: this option doesn't require the application of predictors for each column. Instead, it approximates the covariance for the full dataset. Better performance for datasets that have missing values in many columns.
- **Last Observation Carried Forward (LOFC)**: data replacement
- **Synthetic Minority Oversampling Technique (SMOTE)**: increase the number of underepresented classes in a dataset

To display df as a formatted table: `display(df)`

| | shop | 2020 | 2021 |
|---|---|---|---|
| 0 | ShopX | 34 | 24 |
| 1 | ShopY | 65 | 76 |
| 2 | ShopZ | 48 | 55 |

| | shop | year | value |
|---|---|---|---|
| 0 | ShopX | 2020 | 24 |
| 1 | ShopY | 2020 | 65 |
| 2 | ShopZ | 2020 | 48 |
| 3 | ShopX | 2021 | 25 |
| 4 | ShopY | 2021 | 76 |
| 5 | ShopZ | 2021 | 55 |

melt df: `salesData = pd.melt([salesData], id_vars=shop, value_vars=[2020, 2021])`

Obtain correlation between two df columns: `df.stat.corr('col1', 'col2')`

> K-fold cross-validation

To perform k-fold cross-validation, include the n_cross_validations parameter and set it to a value. This parameter sets how many cross validations to perform, based on the same number of folds.

The best performing fold only identifies the winning pipeline. You discard all the trained models during the k-fold procedure and train the winning pipeline with all available training data to produce the selected model pipeline.

**Pipeline example: train a SVM**

1. Add dataset to experiment (Import Data module, to access data from datasources) -> This step outputs a dataset
2. Add Split Data module to create train/test datasets
3. Add SVM module to initialize SVM classifier
4. Add a trained model
5. Optional: add a permutation feature importance module and connect the trained model and test the dataset
6. Set the metric for measuring performance property to Classification - Accuracy (in this scenario) and run the experiment

**Pipeline example: model must be retrained monthly with new data**

1. Publish the pipeline
2. Retrive the pipeline ID
3. Create a ScheduleRecurrency(frequency='Month', interval=1, start_time='...')
4. Define an AML pipeline schedule using the schedule.create() with the defined recurrence specification

**Pipeline example: obtain AUC:**

1. Train model
2. Score model (get prediction)
3. Evaluate model (compare several predictions)

**Log/monitor the progress of the pipeline execution**

- `pipeline_run.wait_for_completion(show_output=True)`
- Open a new notebook and run `RunDetails(pipeline_run).show()`
- Use the activity log in the azure portal for the ML workspace

**Step reuse**

By default, the step output from a previous pipeline run is reused without rerunning the step unless parameters are changed. Step reuse can reduce the time it takes to run a pipeline, it can lead to stale results, and all steps can be forced to run regardless of individual reuse config.

## 2.4. Training

- To create versions of a model, register new models with the same name. First registered model gets Version 1, each model registered to the same model name increments the version number.
- To include custom metrics during training, add print statements to the scoring script so that the custom information is written to the STDOUT log. Then it will be possible to analyze it using Application Insights.
- For other logging, use logging library for Python or Run.Log()
    - To record metrics (like AUC) in an experiment run, use the `run.log('Accuracy', np.float(acc))` methods

- To register a model ensuring that the PyTorch version can be identified, register the model witha .pt file extension and the default version property.

To obtain the best iteration of the experiment run of AutoML: `automl_run.get_output()[0]` whith returns the best run and the fitted model

The `wait_for_completion()` method will return before all the nodes are created. If the code creates a new cluster compute target, it may be pre-empted due to capacity constraints

**Estimators**

Represents a generic estimator to train data using any supplied framework. To configure an Estimator with a datastore, pass the dataset through scripts_params, to keep the training script independent from azureml-sdk

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

Another way:

```
Estimator(inputs=[file_dataset.as_named_input('training_files').as_mount()])
```

`as.mount()` ensures that data is streamed directly from the source.

**Hyperparameter tuning**

The set of hyperparameter values tried during hyperparameter tuning is known as the search space. The definition of the range of possible values that can be chosen depends on the type of hyperparameter.

Discrete hyperparameters:

- qnormal
- quniform
- qlognormal
- qloguniform

Continuous hyperparameters:

- normal
- uniform
- lognormal
- loguniform

Defining search space, create a params_space with the hyperparameter and the type (choice, normal, uniform). The specific values used in a hyperparameter tuning depend on the type of sampling:

- Grid sampling: only when all hyperparameters are discrete. Tries all possible combinations of parameters
- Random sampling: randomly select a value for each hyperparameter
- Bayesian sampling: selects hyperparameter combinations that will result in improved performance from the previous selection. Can only be used with choice, uniform and quniform. Can't be combined with early-termination policy

**MLflow**

Open source platform for managing the end-to-end machine learning lifecycle. Supports Java, Python, R, REST APIs.

Azure Databricks provides a fully managed and hosted version of MLflow integrated with enterprise security features, high availability, and other Azure Databricks workspace features such as experiment and run management and notebook revision capture.

- Tracking experiments
- Managing and deploying models
- ML packaging
- Model registry
- Model serving

mlflow commands

- `mlflow.log_artifact` method logs file or directory contents.
- `mlflow.log_metric` for metrics

**Model evaluation**

Fairlearn is a Python package to analyze models and evaluate disparity between predictions and prediction performance for sensitive issues. The choice of parity constraint depends on the technique being used and the specific fairness criteria you want to apply.

- **Demographic parity**: in a binary classification scenario, this constraint tries to ensure that an equal number of positive predictions are made in each group
- **True positive rate parity**: minimize disparity in true positive rate across sensitive feature groups
- **False positive rate parity**: minimize disparity in false_positive_rate across sensntive feature groups
- **Equalized odds**: minimize disparity in combined true_positive_rate and false_positive_rate across sensitive feature groups
- **Error rate parity**: Use this constraint with any of the reduction-based mitigation algorithms (Exponentiated Gradient and Grid Search) to ensure that the error for each sensitive feature group does not deviate from the overall error rate by more than a specified amount
- **Bounded group loss**: restrict the loss for each sensitive feature group in a regression model

To see the model in AML designer and from there view its fairness dashboard, run an experiment in which you upload the dashboard metrics from the model, by using the `upload_dashboard_dictionary function`

> Differential provacy

In a differential privacy solution, noise is added to the data when generating analyses so that aggregations are statistically consistent but non-deterministic; and individual contributions to the aggregations cannot be determined.

Low epsilon: reduces the impact of an individual's data on aggregated results, increasing privacy and reducing accuracy.

**Explainers**

- Tabular explainer can explain local and global plus it works with logistic regression. it acts like a wrapper around SHAP explainer algorithms, automatically choosing the one that is most appropriate for your model architecture
- PFI (permutation feature importance) explainer explains the overall behaviour of a model but doesn't explain individual predictions. Analyzes feature importance by shuffling feature values and measuring the impact on prediction performance.
- Mimic explainer: creates a global surrogate model that approximates your trained model and can generate explanations. This explainable model must have the same kind of architecture as your trained model (for example, linear or tree-based). local+global

## 2.5. Deploying a model

- For testing and development, deploying to a local service or compute instance or a container instance (Docker) is good. In this case, no need to input deployment_target
    - Allows testing and debugging without re-deploying to production
- For production, AKS is preferred (better performance, scalability, security). Needs to specify deployment_target

To deploy a model as real-time inferencing service:

1. Register a trained model. as input needs: model name, model path (.pkl file)
2. Define an inference configuration: a script to load the model and return predictions, an environment
    - Create entry/scoring script: init (load model) and run (get predictions from model) functions -> .py file
    - Create environment: use CondaDependencies to create default env (includes azureml-defaultsn numpy, pandas), then serialize the env to a string and save it -> .yml file. Use `from_conda_specification`
        - If env already exists, use `from_existing_conda_specification`
3. Define deployment configuration: configure compute (first create aks cluster)
    - If authentication should be enabled with azure Active Directory, *token_auth_enabled=True*
    - If authentication should be enabled as key-based, *auth_enabled=True*
4. Deploy model

To enable monitoring of the deployed model, enable application insights for the service endpoint and see the logged data in azure portal.

To test the deployed service, endpoint and key are needed:

```
input_json = json.dumps({"data": my_data})
headers = {"Content-Type":"application/json", "Authorization":"Bearer " + key}
response = requests.post(endpoint, input_json, headers=headers)
```

If you already have access to the Webservice:

```
ws = Workspace.from_config()
servce = Webservice(name='mlmodel1-service', workspace=ws)
input_json = json.dumps({"data": [[1, 2, 3], [3, 4, 5]]})
predictions = service.run(input_json)
```

- Local importance: influence of features on a specific prediction
- Global importance: overall indication of feature influence

# 3. deep learning concepts

## Regression

- Necessary data: dataset with historical **features** of the entity to be predicted + known **label** values to train model.
- Metrics for regression
  - RMSE (root mean square error), the lower the better
  - R-Squared value, the higher the better. Measures the variance between predicted and actual values
- p-value is used to assess if model coefficients are statistically significant. if p-value is 0.05, there's 5% chance of seeing the correlation between input and output by mere chance (good)

## Clustering

To infer a clustering model, don't "score model", but "assign data to clusters".

In k-means training (supervised),

- Centroids don't change between iterations
- The residual sum of squares (RSS) goes lower
- The algorithm needs a fixed number of iterations

## Classification

AUC (Area under the Receiver Operating Characteristic Curve) is a metric for classification. for imbalanced dataset, AUC_weighted. AUC=0.5 means random guessing

## Feature scoring statistical methods

To calculate the correlation between features and output label

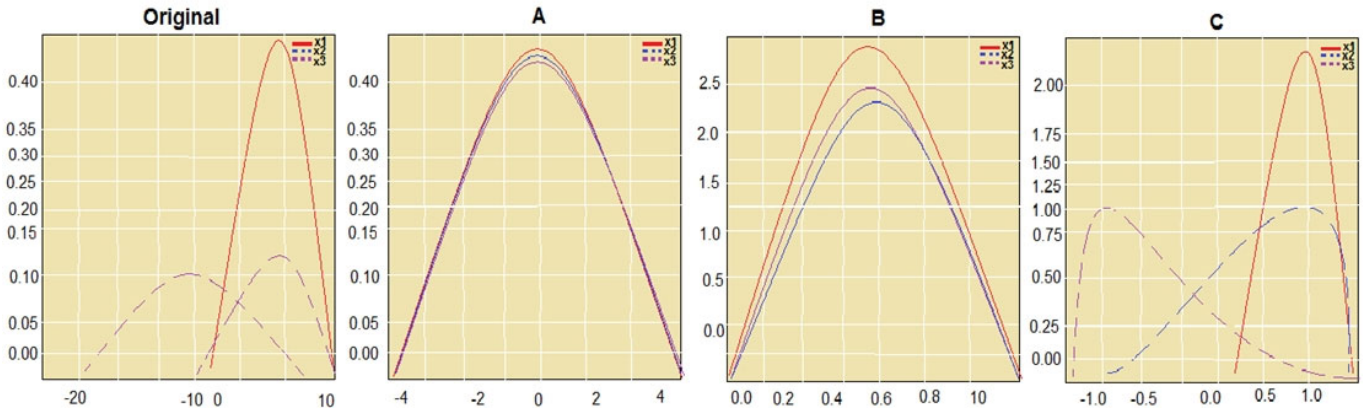
- Chi squared: features and label can be text or numeric
- Pearson correlation: features must be numeric, label can be text/numeric

- Spearman correlation: features must be numeric, label can be text/numeric

## Azure cognitive services

Azure cognitive services has pre-trained models and little room for customization. Works with video, audio and text.

## Scalers




- A: Standard scaler
- B: MinMax scaler
- C: Normalizer scaler