DUMPSARENA

Databricks Certified Machine Learning Associate Exam

Databricks Databricks-Machine-Learning-Associate

Version Demo

Total Demo Questions: 10

Total Premium Questions: 74

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

A data scientist has produced two models for a single machine learning problem. One of the models performs well when one of the features has a value of less than 5, and the other model performs well when the value of that feature is greater than or equal to 5. The data scientist decides to combine the two models into a single machine learning solution.

Which of the following terms is used to describe this combination of models?

A. Bootstrap aggregation

B. Support vector machines

C. Bucketing

D. Ensemble learning

E. Stacking

ANSWER: D

Explanation:

The best term for combining multiple models into a single predictive solution is **ensemble learning**. In the scenario, the data scientist is effectively using different “experts” for different regions of the feature space (feature < 5 vs. feature ≥ 5) and combining them into one overall system. That is an ensemble concept: multiple models are used together to improve robustness and performance compared to relying on only one model.

While **stacking** is a specific type of ensemble (using a meta-model to learn how to combine base model predictions), the prompt doesn’t mention a meta-learner or training a combiner on model outputs—just that the two models are combined into one solution. **Bootstrap aggregation (bagging)** is another specific ensemble method, but it refers to training many models on bootstrapped samples and aggregating their predictions, which is not what’s described. **Bucketing** is a feature engineering/binning concept, not a model-combination strategy. **Support vector machines** is a single model family, not a combination approach.

References: scikit-learn: Ensemble methods, Ensemble learning (overview).

QUESTION NO: 2

A machine learning engineer has grown tired of needing to install the MLflow Python library on each of their clusters. They ask a senior machine learning engineer how their notebooks can load the MLflow library without installing it each time. The senior machine learning engineer suggests that they use Databricks Runtime for Machine Learning.

Which of the following approaches describes how the machine learning engineer can begin using Databricks Runtime for Machine Learning?

A. They can add a line enabling Databricks Runtime ML in their init script when creating their clusters.

B. They can check the Databricks Runtime ML box when creating their clusters.

C. They can select a Databricks Runtime ML version from the Databricks Runtime Version dropdown when creating their clusters.

D. They can set the runtime-version variable in their Spark session to œml .

ANSWER: C

Explanation:

Databricks Runtime for Machine Learning (often called “Databricks Runtime ML”) is a pre-packaged Databricks runtime image that comes with common ML and DL libraries already installed, including MLflow. To start using it, you don’t “enable” it via an init script or a checkbox; you choose an ML runtime as the cluster’s runtime. Concretely, when creating (or editing) a cluster, you select a runtime whose name indicates it is the ML variant (for example, “Databricks Runtime ML x.y”). Once the cluster starts with that runtime, notebooks attached to the cluster can import and use MLflow without manually installing it each time.

Option C correctly describes this: selecting a Databricks Runtime ML version from the runtime dropdown during cluster creation. Option A is incorrect because init scripts are for custom bootstrapping, not switching the base runtime distribution. Option B is incorrect because the UI does not provide a generic “Databricks Runtime ML box”; the selection is done via the runtime version dropdown. Option D is incorrect because you cannot change the cluster runtime by setting a Spark session variable.

References: Databricks: Create a cluster, Databricks Runtime for Machine Learning

QUESTION NO: 3

A machine learning engineer has been notified that a new Staging version of a model registered to the MLflow Model Registry has passed all tests. As a result, the machine learning engineer wants to put this model into production by transitioning it to the Production stage in the Model Registry.

From which of the following pages in Databricks Machine Learning can the machine learning engineer accomplish this task?

A. The home page of the MLflow Model Registry

B. The experiment page in the Experiments observatory

C. The model version page in the MLflow Model Registry

D. The model page in the MLflow Model Registry

ANSWER: C

Explanation:

The correct place to transition a specific registered model *version* from Staging to Production is the **model version page** in the MLflow Model Registry. Stage transitions (e.g., Staging → Production, or archiving older versions) are actions applied to an individual version, and the version details UI is where Databricks exposes the “Stage” control for that version along with its metadata (run link, metrics/params, comments, and permissions). This aligns with the Model Registry workflow: you select the registered model, then select the exact version you want, and then transition its stage.

Option A is too generic: the registry home page is primarily for browsing/searching registered models, not performing version-specific stage transitions. Option B is incorrect because experiments track runs and artifacts; they don’t manage registry stages for registered model versions. Option D (the model page) is close, but the actual stage transition is performed at the version level; you typically navigate from the model page into a specific version page to change its stage.

References: Databricks MLflow Model Registry, MLflow Model Registry documentation.

QUESTION NO: 4

A data scientist has been given an incomplete notebook from the data engineering team. The notebook uses a Spark DataFrame spark_df on which the data scientist needs to perform further feature engineering. Unfortunately, the data scientist has not yet learned the PySpark DataFrame API.

Which of the following blocks of code can the data scientist run to be able to use the pandas API on Spark?

A. import pyspark.pandas as ps df = ps.DataFrame(spark_df)

B. import pyspark.pandas as ps df = ps.to_pandas(spark_df)

C. spark_df.to_sql()

D. import pandas as pd
df = pd.DataFrame(spark_df)

E. spark_df.to_pandas()

ANSWER: A

Explanation:

The pandas API on Spark (formerly “pandas-on-Spark” / Koalas) lets you work with a pandas-like API while still executing on Spark. To do that, you must create a pandas-on-Spark DataFrame (not a local pandas DataFrame). The correct pattern is to import `pyspark.pandas` (or `pyspark.pandas` in older runtimes) and convert the existing Spark DataFrame using the pandas API on Spark constructor. Option A does exactly this by creating a pandas-on-Spark DataFrame from `spark_df`, enabling familiar pandas-style operations that are translated into Spark execution.

Option B is incorrect because `ps.to_pandas` is not the standard conversion function in pandas API on Spark; “to_pandas” typically refers to collecting data into a local pandas DataFrame (which defeats the purpose). Option C is unrelated (Spark DataFrames don’t provide `to_sql()` as a general conversion to pandas API on Spark). Option D creates a local pandas DataFrame and won’t work directly from a Spark DataFrame without collecting, and it won’t use the pandas API on Spark. Option E (`spark_df.toPandas()`) collects all data to the driver as a local pandas DataFrame, which is not the pandas API on Spark.

References: Apache Spark: pandas API on Spark, PySpark pandas API reference

QUESTION NO: 5

An organization is developing a feature repository and is electing to one-hot encode all categorical feature variables. A data scientist suggests that the categorical feature variables should not be onehot encoded within the feature repository.

Which of the following explanations justifies this suggestion?

A. One-hot encoding is not supported by most machine learning libraries.

B. One-hot encoding is dependent on the target variable's values which differ for each application.

C. One-hot encoding is computationally intensive and should only be performed on small samples of training sets for individual machine learning problems.

D. One-hot encoding is not a common strategy for representing categorical feature variables numerically.

E. One-hot encoding is a potentially problematic categorical variable strategy for some machine learning algorithms.

ANSWER: E

Explanation:

Option E is correct because one-hot encoding is a modeling/algorithm-dependent transformation and can be a poor default to bake into a shared feature repository. A feature store is meant to provide reusable, generally applicable features across many downstream models and teams. If you one-hot encode at the repository level, you lock in a specific representation that may be suboptimal (or even harmful) for some algorithms and workflows. For example, tree-based models often don't need one-hot encoding and can suffer from unnecessary dimensionality expansion; high-cardinality categoricals can explode the number of columns, increasing storage, compute, and operational complexity. Also, different consumers may want different encodings (one-hot vs. hashing vs. target encoding vs. embeddings), so keeping the raw categorical feature in the repository preserves flexibility and lets each training pipeline apply the appropriate encoding consistently with its model and validation strategy.

Why others are wrong: A is false because one-hot encoding is widely supported (e.g., Spark ML OneHotEncoder). B is false because one-hot encoding is not target-dependent (that's target encoding). C is misleading: it can be computationally heavier, but it's not something you "only" do on small samples. D is false because one-hot encoding is a very common strategy.

References: Spark ML OneHotEncoder, Databricks Feature Store overview.

QUESTION NO: 6

An organization is developing a feature repository and is electing to one-hot encode all categorical feature variables. A data scientist suggests that the categorical feature variables should not be onehot encoded within the feature repository.

Which of the following explanations justifies this suggestion?

A. One-hot encoding is a potentially problematic categorical variable strategy for some machine learning algorithms.

B. One-hot encoding is dependent on the target variables values which differ for each apaplication.

C. One-hot encoding is computationally intensive and should only be performed on small samples of training sets for individual machine learning problems.

D. One-hot encoding is not a common strategy for representing categorical feature variables numerically.

ANSWER: A

Explanation:

The best justification is that one-hot encoding can be a poor "one-size-fits-all" representation to store in a shared feature repository because it may be suboptimal (or unnecessary) for some downstream models and use cases. One-hot encoding expands a single categorical column into many sparse indicator columns, increasing feature dimensionality and storage/compute costs. It can also create maintenance issues when categories evolve (new levels require schema changes) and can be redundant for models that can natively handle categorical features or benefit from alternative encodings (e.g., embeddings, target encoding, hashing). In a feature store/repository, the goal is typically to publish reusable, stable, model-agnostic features; leaving categoricals in a canonical form and performing model-specific encoding in the training/inference pipeline is often the more flexible design.

Option A captures this idea: one-hot encoding can be problematic for some algorithms and is not universally the right choice to bake into shared features. Option B is incorrect because one-hot encoding is not target-dependent (that's more relevant to target encoding). Option C is too absolute; one-hot encoding can be used at scale, though it may be expensive. Option D is false because one-hot encoding is a very common categorical encoding strategy.

References: Databricks Feature Store, Apache Spark ML OneHotEncoder

QUESTION NO: 7

Which of the following evaluation metrics is not suitable to evaluate runs in AutoML experiments for regression problems?

A. F1

B. R-squared

C. MAE

D. MSE

ANSWER: A

Explanation:

For regression problems, AutoML evaluates models using regression metrics such as R-squared (R^2), mean absolute error (MAE), mean squared error (MSE), and related variants (for example RMSE). These metrics measure how close predicted continuous values are to the true continuous targets. In contrast, **F1** is a classification metric (the harmonic mean of precision and recall) and requires discrete class labels (or a thresholded probability output). Because regression outputs are continuous and do not naturally define true/false positives/negatives, F1 is not an appropriate metric for evaluating regression runs in Databricks AutoML.

Option B (R-squared) is a standard goodness-of-fit metric for regression. Option C (MAE) and option D (MSE) are also standard regression loss/error metrics and are commonly used to compare regression models. Therefore, the only metric in the list that is not suitable for regression evaluation is F1.

References: Databricks AutoML documentation, scikit-learn: Model evaluation

QUESTION NO: 8

A machine learning engineer wants to parallelize the inference of group-specific models using the Pandas Function API. They have developed the apply_model function that will look up and load the correct model for each group, and they want to apply it to each group of DataFrame df.

They have written the following incomplete code block:

```
prediction_df = (df
    .groupby("device_id")
    .______(apply_model, schema=apply_return_schema)
)
```

Which piece of code can be used to fill in the above blank to complete the task?

A. applyInPandas

B. groupedApplyInPandas

C. mapInPandas

D. predict

ANSWER: A

Explanation:

The correct choice is **applyInPandas**. In Spark, the Pandas Function API for grouped operations is done by calling `df.groupBy(...).applyInPandas(func, schema)`. This executes your Python function once per group, where the function receives a pandas DataFrame for that group and returns a pandas DataFrame matching the provided schema. That's exactly what you want for "group-specific models": group by the key (for example, device_id), then apply a function that loads the appropriate model for that group and produces predictions, all in parallel across Spark tasks.

groupedApplyInPandas is not the correct API name for modern PySpark; the supported method is `applyInPandas` on a grouped DataFrame. **mapInPandas** applies an iterator-of-pandas-DataFrames function to partitions (not groups), so it won't guarantee group isolation. **predict** is not a Spark DataFrame method for this use case.

References: PySpark GroupedData.applyInPandas, Databricks Pandas Function API

QUESTION NO: 9

A data scientist is using MLflow to track their machine learning experiment. As a part of each of their MLflow runs, they are performing hyperparameter tuning. The data scientist would like to have one parent run for the tuning process with a child run for each unique combination of hyperparameter values. All parent and child runs are being manually started with mlflow.start_run.

Which of the following approaches can the data scientist use to accomplish this MLflow run organization?

A. They can turn on Databricks Autologging

B. They can specify nested=True when starting the child run for each unique combination of hyperparameter values

C. They can start each child run inside the parent run's indented code block using mlflow.start runO

D. They can start each child run with the same experiment ID as the parent run

E. They can specify nested=True when starting the parent run for the tuning process

ANSWER: B

Explanation:

To create a single "tuning" parent run with one child run per hyperparameter combination, MLflow requires you to explicitly start nested runs. The correct pattern is: start the parent run normally (no `nested=True`), then for each trial start a new run with `mlflow.start_run(nested=True)`. This sets the child run's `mlflow.parentRunId` tag to the active parent run, producing the desired hierarchy in the MLflow UI and keeping all trials grouped under the tuning run.

Option A (autologging) can log parameters/metrics automatically, but it does not by itself create a parent/child run structure for manual tuning loops. Option C is incorrect because simply placing `start_run` "inside" a code block doesn't make it a child; without `nested=True`, MLflow will end the active run or raise an error depending on context. Option D is wrong because experiment ID only determines which experiment the run belongs to, not parent-child relationships. Option E is wrong because `nested=True` is intended for the child run; the parent run should be a regular run.

References: MLflow Tracking: Nested Runs, mlflow.start_run API.

QUESTION NO: 10

A data scientist is performing hyperparameter tuning using an iterative optimization algorithm. Each

evaluation of unique hyperparameter values is being trained on a single compute node. They are performing eight total evaluations across eight total compute nodes. While the accuracy of the model does vary over the eight evaluations, they notice there is no trend of improvement in the accuracy. The data scientist believes this is due to the parallelization of the tuning process.

Which change could the data scientist make to improve their model accuracy over the course of their tuning process?

A. Change the number of compute nodes to be half or less than half of the number of evaluations.

B. Change the number of compute nodes and the number of evaluations to be much larger but equal.

C. Change the iterative optimization algorithm used to facilitate the tuning process.

D. Change the number of compute nodes to be double or more than double the number of evaluations.

ANSWER: A

Explanation:

The issue is that many “iterative” hyperparameter optimization methods (for example Bayesian optimization / TPE in Hyperopt) learn from results of earlier trials to pick better hyperparameters in later trials. If you run all 8 trials fully in parallel (8 evaluations on 8 nodes at once), the optimizer can’t incorporate feedback from earlier trials because there aren’t any earlier completed trials yet—so the search behaves more like random search, and you won’t see a consistent improvement trend across trial order. A practical fix is to reduce parallelism so some trials complete and inform subsequent ones (for example, use fewer workers than total trials, or run more trials than workers so there are multiple “waves” of trials). That change directly addresses the data scientist’s suspicion about parallelization limiting the optimizer’s sequential learning.

Option A captures this: using half (or less) as many compute nodes as evaluations forces sequential batches, enabling the optimizer to adapt and typically improving the chance of finding better hyperparameters over time. Option B keeps everything parallel, so it doesn’t restore sequential feedback. Option C might help in general, but it doesn’t specifically solve the stated parallelization problem—most iterative optimizers still need some sequential signal. Option D is not feasible/meaningful (more nodes than evaluations) and doesn’t help.

References: Databricks: Hyperparameter tuning with Hyperopt, Hyperopt documentation