# [Proctor Guide]

# Challenge 02 – Ingest and Prepare Data

## Summary

A complete set of Databricks notebooks that covers Challenge02 is available with this documentation in the Solutions folder.

Refer to these as needed to help teams accomplish the tasks in this challenge. The notebooks are broken out per task.

## Task 1 – Copy source data files to your Azure storage location

Tools they could use include Azure Storage Explorer, azcopy, az cli, etc.

Note: at the time of this writing (January 2019), Azure Storage Explorer and azcopy do NOT support copying from a v2 storage account (where the source files are located) into a v2 storage account that is also ADLS v2-enabled. If attendees cannot see a “Paste” button in Storage Explorer when trying to get the source data files, have them verify whether their storage account is ADLS v2-enabled, and if so, they should create a new storage account without ADLS v2.

Workaround to copy files into ADLS v2 (NOTE: This method does not work for hierarchy-enabled v2 storage):   
Create v1-enabled storage container and copy the source files into the v1 container. This can be done in Storage Explorer copy/paste. Keep in mind, it will take some time for the files to copy over; this does not happen immediately as it appears (folders/files will appear, but double-check file sizes to verify the copy completed). Then copy from the v1 storage container into a v2 storage container.

## Task 2 – Connect cluster to your Azure storage location

There are two ways to mount an Azure storage container in Databricks:

1. Using the DataFrame API. See <https://docs.azuredatabricks.net/spark/latest/data-sources/azure/azure-storage.html#access-azure-blob-storage-using-the-dataframe-api>. This method is very straightforward and is probably preferable in a time-limited hackathon where attendees may have limited Databricks/Azure experience.
2. Using DBFS. See <https://docs.azuredatabricks.net/spark/latest/data-sources/azure/azure-storage.html#mount-azure-blob-storage-containers-with-dbfs>.  
   The samples shown in the docs specify use of secrets stored in a Databricks secret scope, which requires either an Azure Key Vault (AKV) or a Databricks backing. The AKV backing can be created in the Azure portal, and the Databricks backing requires the Databricks CLI, which is a separate install.  
   Using AKV, the sequence is to create the AKV first, note pertinent configuration details, create a Secret (not a Key!) there, then create the Databricks secret scope, where the AKV details must be provided, and noting pertinent Databricks secret scope details which will then be pasted into the sample code provided in the doc linked above (Using DBFS). See <https://docs.azuredatabricks.net/user-guide/secrets/secret-scopes.html#secret-scopes>.   
   This method is less straightforward. If teams get stuck, either recommend they use method 1, or walk through the steps on the secret-scopes link in detail. The procedure does work, but for teams with limited Databricks or Azure experience, this method can be more time-consuming and error-prone than method 1.

IMPORTANT: as attendees connect their Databricks cluster to Azure storage and create mount point(s), the storage container(s) to which attendees will attach mount point(s) must exist already. If attendees get error messages when trying to create mount points, have them double-check that the container(s) they are connecting to exist(s), and that they have correctly copy/pasted keys, SAS URIs, etc.

## Task 3 – Create a Hive Database

Please refer 03-Prep-Database.scala

## Task 4 – Create Hive Tables for Reference Data

Teams will need to take the following steps:

1. Understand the source file schemas
2. Read each source file into a dataframe
3. Write each dataframe out to Parquet format
   1. This is needed to remove header rows from CSV files, and because Parquet files are faster and more efficient than CSV for Hive operations
4. Create an external Hive table on top of each Parquet output location

Note! Parquet reads and writes can be parallelized and partitioned. This means that a single dataframe can result in multiple Parquet files. Teams should be very careful to write each dataframe out to its own dedicated destination folder!

## Task 5 – Create Hive Tables for Transaction Data

Following a similar process as in task 4, teams should wind up with one Hive table for Yellow, and one for Green.

To succeed, they will need to do the following for each of Yellow and Green:

1. Understand how the source data file schemas change over the years
2. Create an external Hive table with a single schema that covers all the source year schemas
   1. They will need to specify a file system location for this table. The folder they specify should be the root folder to which they will write Parquet outputs. Each of the two Hive tables (yellow, green) should have its own root folder.  
      Examples: /mnt/data/dtafy19/processed/yellow-taxi/ and /mnt/data/dtafy19/processed/green-taxi/
3. Iterate through all the source data folders (for each year, for each month) and:
   1. Using the correct source schema for that year/month, read the source data into a dataframe
   2. Adjust the dataframe to confirm to the single schema of the target Hive table created above
   3. Write output as Parquet, into a new folder underneath the Hive table’s root folder. Example:  
      /mnt/data/dtafy19/processed/yellow-taxi/trip\_year=2016/trip\_month=10/
   4. Consider encouraging teams to partition the data for each year/month to improve query performance.

## Task 6 – Conform and Merge Data Sets

In this task, teams will create one merged master from the yellow and green merged data sets created in task 5.

To do this, they will need to ensure both data sets have the same schema, then union them into one data set and write that out to Parquet, then create a Hive external table on that final Parquet data.

Teams have successfully completed challenge 2 when they can query one Hive table that contains ALL yellow and green data.

## Bonus Task – Connect Databricks Notebook(s) to github

The Databricks documentation for how to do this is here: <https://docs.azuredatabricks.net/user-guide/notebooks/github-version-control.html>

Note that the github UI has changed since the Databricks doc was written. Attendees should now go to github Settings 🡪 Developer Settings 🡪 Personal access tokens and proceed per the Databricks documentation.

If attendees create a new github repo, they should first clone it to their machine and commit a file (e.g. a readme or a .gitignore) and commit/push that. Without an initial commit, git sync in Databricks may fail, but succeed after new github repo is initialized.