



Exploring Apache Spark and Spark SQL in Microsoft Azure Databricks



Introduction

This class introduces students to Apache Spark on Azure with Databricks. It helps student to understand the value proposition of Apache Spark over other Big Data technologies like Hadoop. They should understand the similarities between Hadoop & Spark, their differences and respective nuances. They should be able to decide when to use what and why for a given business use case in a typical enterprise environment.

Azure specific highlights of Apache Spark

Source: <https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks>

#1 Ease creation

You can create a new Spark cluster in minutes without the complexity usually associated with infrastructure

#2 Ease of use

Spark cluster in Databricks give you access to a notebook interface. You can use these notebooks for interactive data processing and visualization.

#3 REST APIs

Spark clusters in Databricks allows you to connect to a REST API and work with the data you produced during your analysis. On top of that, Databricks provides an interface allowing you to schedule and monitor your Spark jobs.

#4 Support for Azure Data Lake Storage

Spark clusters in Databricks can use Azure Data Lake Storage as both the primary storage or additional storage.

#5 Integration with Azure services

Spark cluster in Databricks comes with a connector to Azure Event Hubs. You can build streaming applications using the Event Hubs, in addition to Apache Kafka, which is already available as part of Spark.

#6 Support for ML Server

Databricks ML Flow platform speeds up ML development and deployment. Also, the notebooks support using Python which is the de facto language for ML life cycle.

#7 Integration with Azure DevOps

In a real world utilization of Databricks and Spark, CI/CD processes become important. As part of Azure, Databricks works nicely with AzureDevops and allow you to manage code versioning as well as Continuous delivery.

#8 Scalability

Databricks allows you to seamlessly replace smaller cluster by bigger cluster and re-attach your notebooks to the new cluster in minutes. Price will increase with cluster usage and size.

Main highlights of Spark SQL

Source: <http://spark.apache.org/sql/>

#1 Integrated - Seamlessly mix SQL queries with Spark programs. Spark SQL lets users query structured data inside Spark programs, using either SQL or a familiar DataFrame API. Usable in Java, Scala, Python and R.

#2 Uniform Data Access - Connect to any data source the same way. DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. Users can even join data across these sources.

#3 Hive Compatibility - Run unmodified Hive queries on existing data. Spark SQL reuses the Hive frontend and metastore, giving users full compatibility with existing Hive data, queries, and UDFs.

#4 Standard Connectivity - Connect through JDBC or ODBC. A server mode provides industry standard JDBC and ODBC connectivity for business intelligence tools.

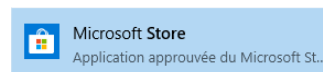
Takeaways

1. Provision an Databricks Spark Cluster.
2. Access data from Azure storage container and create Dataframe.
3. Understand joins, functions and user defined functions.
4. Connect your Databricks Spark Cluster with Power BI Visualization.

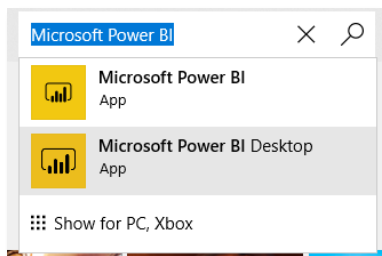
Prerequisites

- a) An Azure subscription. [See here](#).
- a) Microsoft Power BI Desktop [See here](#)

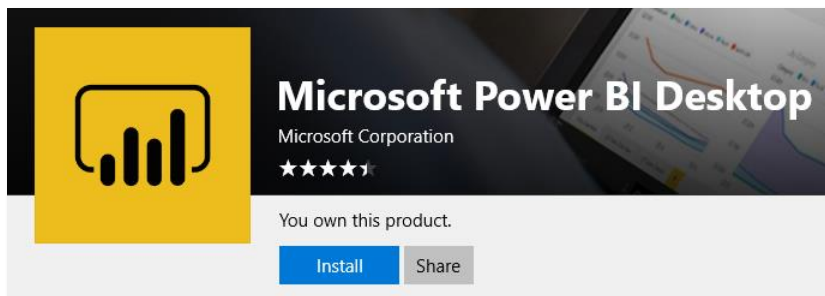
- 1) Launch the Microsoft Store (from windows 10)



- 2) In the Search bar, type Microsoft Power BI Desktop and select Microsoft Power Bi Desktop.



- 3) Click on **Install**



Section 1 - Prepare Cluster and dataset

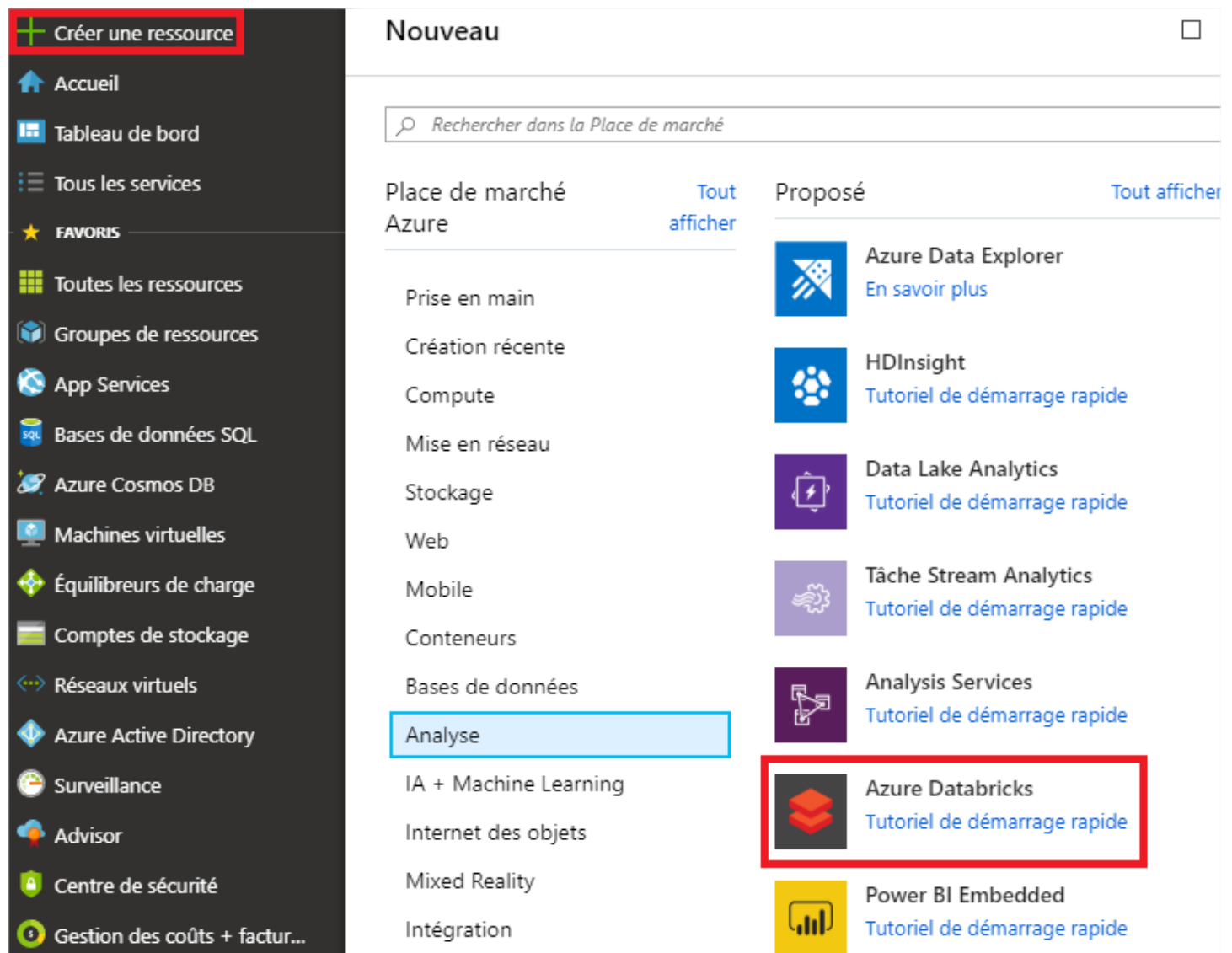
Provision an Azure Databricks cluster

Access Azure Portal

1. Sign in to the [Azure portal](#).

Create Azure Databricks cluster

1. Click Create new resource, click Analysis, and then click Azure Databricks.



Provide Cluster Details

1. In the Azure Databricks Service blade, enter an available **Workspace Name**. Note that it cannot include "Microsoft" or "MS".

Azure Databricks Service

* Workspace name

SparkAgileDSS

✓

* Abonnement ⓘ

Visual Studio Enterprise avec MSDN

▼

* Groupe de ressources ⓘ

☒ Créer nouveau

☐ Utiliser existant

admin_spark

✓

* Emplacement

Est des États-Unis 2

▼

* Pricing Tier ([View full pricing details](#))

Standard (Apache Spark, Secure with Azur... ▼

Deploy Azure Databricks workspace in your Virtual Network ([preview](#))

☐ Yes

☒ No

Créer

Options d'automatisation

A green check mark appears beside the cluster name if it is available.

2. For **Subscription**, if you have more than one subscription, click the Subscription entry to select the Azure subscription to use for the cluster.

Provision cluster

1. Click **Create** button to finalize cluster creation. This may take 5 minutes.

This creates the cluster and adds a tile for it to the **Startboard** of your Azure portal.

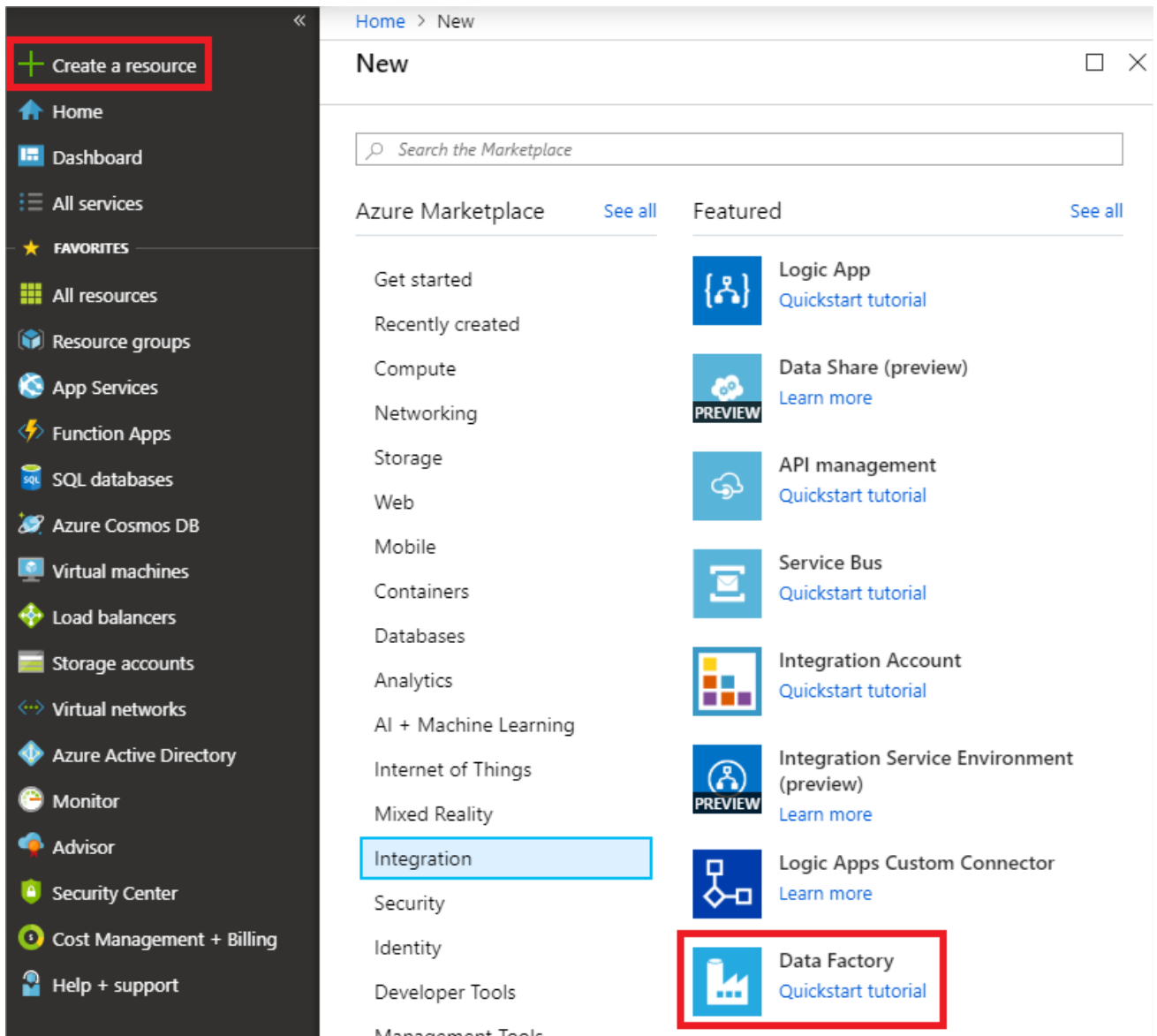
The screenshot shows the SparkAgileDSS Azure Databricks Service interface. The sidebar on the left contains navigation links: Overview, Journal d'activité, Contrôle d'accès (IAM), Étiquettes, Paramètres, Virtual Network Peerings, Verrous, Exporter le modèle, Support + dépannage, and Nouvelle demande de support. The main content area displays the cluster details for 'admin_spark'. It includes a 'Delete' button, a link to 'Groupe de ressources (modifier) admin_spark', a link to 'Abonnement (modifier) Visual Studio Enterprise avec MSDN', the 'ID d'abonnement ae6e4f30-ecd0-4f79-86d5-781ac8351e5f', the 'Managed Resource Group databricks-rg-SparkAgileDSS-znmenei4ld6jy', the 'URL https://eastus2.azuredatabricks.net', and the 'Pricing Tier standard'. A large red 'Launch Workspace' button is centered below the details. At the bottom, there are six tiles: Documentation, Getting Started, Import Data from File, Import Data from Azure Storage, Notebook, and Admin Guide.

Load datasets files to storage account.

In this section, you'll copy the files required for the lab to the storage account previously created. You'll copy the files between two storage accounts with the help of Data Factory.

To copy the files, follow the below steps.

1. Create a Data Factory instance from Azure Portal.



2. Fill the required information to create the new **Data Factory**. Enter a name you will easily recognize. Choose your subscription, created with your account and include the **Data Factory** in the resource group we created with the Databricks cluster. We will use the V2. Make sure you choose the same geographic area so you won't be charge for getting data out of the data center.

Home > New > New data factory

New data factory □ ×



* Name ⓘ
agiledssSparkLab ✓

* Subscription
Visual Studio Enterprise avec MSDN ▼

* Resource Group ⓘ
☐ Create new ☒ Use existing
lab_spark_agiledss ▼

Version ⓘ
V2 ▼

* Location ⓘ
East US ▼

 Integrate with GIT source control to do collaboration, source control, change tracking, change difference, continuous integration and deployment etc 

☐ Enable GIT ⓘ

* GIT URL ⓘ
`https://github.com/michaelbond`

* Repo name ⓘ

* Branch Name ⓘ
master

* Root folder ⓘ
/

3. In your Portal Home, select your **Data Factory**.

Azure services [See all \(+100\) >](#)

Virtual machines Storage accounts App Services SQL databases Azure Database for PostgreSQL Azure Cosmos DB

Make the most out of Azure

Learn Azure with free online courses by Microsoft
[Microsoft Learn >](#)

Monitor your apps and infrastructure
[Azure Monitor >](#)

Secure your apps and infrastructure
[Security Center >](#)

Recent resources [See all your recent resources >](#) [See all your resources >](#)

NAME	TYPE	LAST VIEWED
labsparkdf	Data factory (V2)	11 min ago
mtl375	Storage account	16 min ago
agiledss	Storage account	18 min ago

- Once inside the **Data Factory** interface, click on **Author and Monitor**.

labsparkdf
Data factory (V2)

Search (Ctrl+/)

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings
Locks General

Delete

Resource group (change) : [lab_spark_agiledss](#) Type :
Status : Succeeded Getting started :
Location : East US
Subscription (change) : [Visual Studio Enterprise avec MSDN](#)
Subscription ID : ae6e4f30-ecd0-4f79-86d5-781ac8351e5f

Documentation **Author & Monitor**

- Click on Copy Data.

Azure Data Factory

Let's get started

Create pipeline Create pipeline from template **Copy Data** Configure SSIS Integration Set up Code Repository

- Give a name to the pipeline. We will only run it once. Then click **Next**.

Copy Data

1 Properties

2 Source

3 Destination

Connection

Dataset

Connection

Dataset

Properties

Enter name and description for the copy data task.

Task name *

AirlineData_LabSpark

Task description

Task cadence or Task schedule

☒ Run once now ☐ Run regularly on schedule

7. Select the **Azure** tab, then search for **blob**, then click on **Create new connection**. A new window will appear on the top of where you can select **Azure Blob Storage**. Once this is done, click on **Continue** at the bottom right of the screen.

Copy Data

1 Properties

2 Source

3 Destination

4 Settings

5 Summary

6 Deployment

Source data store

Specify the source data store for the copy task. You can use an existing data store connection or specify a new one.

All **Azure** Database File Generic protocol NoSQL Services and apps

blob

+ Create new connection

New Linked Service

Search

All **Azure** Database File Generic protocol NoSQL Services and apps

Azure Blob Storage

Azure Cosmos DB (MongoDB API)

Azure Cosmos DB (SQL API)

Azure Data Explorer (Kusto)

Azure Data Lake Storage Gen1

Azure Data Lake Storage Gen2

8. Give a name to your source, then select the default Runtime. In the **Authentication method** box, select **Account key**, then **Connection String**, then **Enter manually**. In the fields below, the **Storage account name** is agiledss and the key is :
6YJEwcCQZarYJAYwcWj5l/kGs/A0evANjeqE7UE/Kfb0ig3c603z4AF9PfdVsWAWoSg8Pcj23T6Gw
khoOi+bLw==. Test the connection and click **Finish**.

←

New Linked Service (Azure Blob Storage)

×

Name *

AirlineInfoSource

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Authentication method

Account key

Connection String

Azure Key Vault

Account selection method

From Azure subscription

Enter manually

Storage account name *

agiledss

Storage account key

Azure Key Vault

Storage account key *

Endpoint Suffix

Additional connection properties

+ New

Annotations

+ New

Advanced

- Clear the filter and select the newly created Data Source, then click Next.

Source data store

Specify the source data store for the copy task. You can use an existing data store or create a new one.

All

Azure

Database

File

Generic protocol

NoSQL

Filter by name

+ Create new connection

AirlineInfoSource

- Select the browser **sampledata**, then click **Next**.

Choose the input file or folder

Select a source folder or file to be copied to the destination data store.

File or folder *

Binary copy ☐

Compression type

Copy file recursively ☒

Filter by last modified

- On the next screen, click **Detect Text Format**, then **Next**.

File format settings

File format [Detect Text Format](#)

Column delimiter ☐ Use custom delimiter

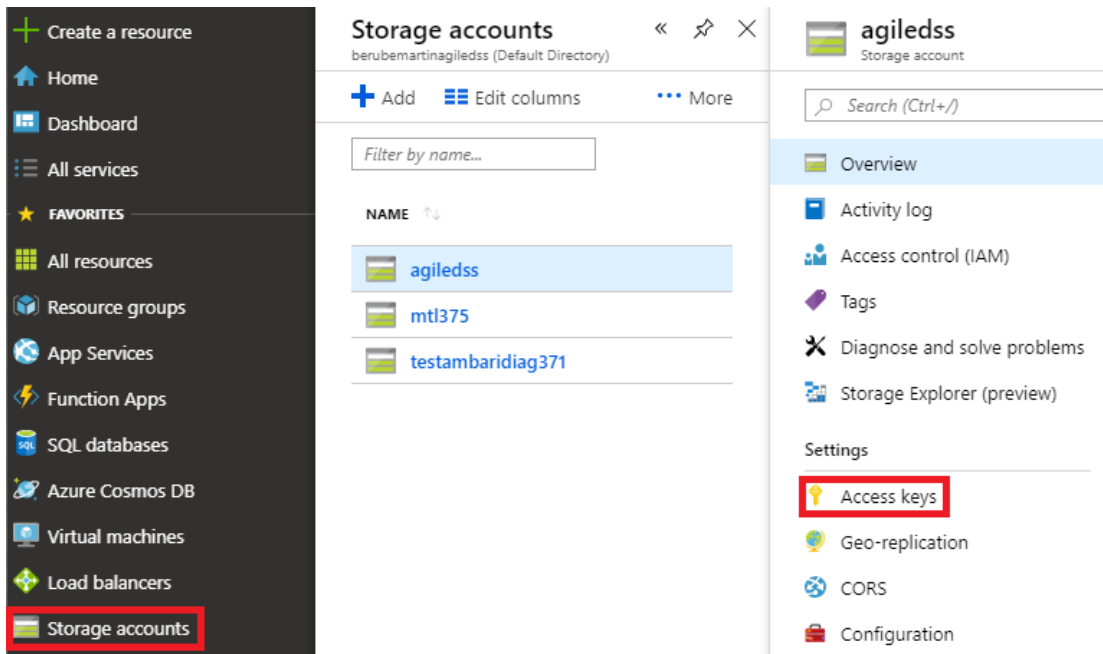
Row delimiter ☐ Use custom delimiter

Skip line count

☐ Column names in the first row

▶ Advanced

- We will now create a new connection to deposit the data in the Container you create. Click on **Create new connection**, then select **Azure Blob Storage** just like we did at Step 7.
- Enter a name for your destination and select the same elements as Step 8 until **Enter manually**. To the remaining fields you will need to go to your Azure Portal window, click on **Storage Accounts**, then select the one you created. Finally, under **Access keys**, copy one of the key strings (any of the 2 ending with “==” should do). Paste it in the **Storage account key** and fill your **Storage account name** with the from your container account. Test connection, then **Finish**.



14. Choose a Folder where you want to save the data or create one. Compress it to Gzip and Fastest. Leave the format to default **Text format**, it's the csv type we want.

Choose the output file or folder

Specify a folder that will contain output files or a specific output file in the destination data store.

Folder path *

File name

Compression type

Compression level

Copy behavior

15. Click **Next** to Settings and Summary. In the Deployment screen, click on **Monitor**. This should be quick and once it's completed, go check in your **Container** to make sure the files have been copied.
16. In Azure portal, navigate to your storage account, then Containers below 'BLOB SERVICE' (see following screenshot), and verify that a new container 'sparklabdata' has been created, containing all the resources:

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Access keys

Configuration

Shared access signature

Metrics (preview)

Properties

Locks

Automation script

BLOB SERVICE

Containers

CORS

Custom domain

Status

Primary: Available

Location

East US 2

Subscription (change)

Visual Studio Ent...

Subscription ID

aa92050a-a246-4...

Search containers by prefix

NAME

☒

sparklabdata...

Search blobs by prefix (case-sensitive)

NAME

MODIFIED

Flight

References

Section 2 - Spark SQL and Dataframe

Access data from Azure storage container and Create Data frame.

Access Azure

1. Sign in to the [Azure portal](#).
2. Click tile for your Spark Service.

Toutes les ressources
agiledss (Default Directory)

+ Ajouter ≡ Modifier les colonnes ↻ Actualiser ↓ Exporter au format CSV ⬇ Attribuer des étiquettes 🗑 Supprimer ↺ Essayer la préversion

Abonnements : Visual Studio Enterprise avec MSDN – Vous ne voyez pas d'abonnement ? [Ouvrir les paramètres de répertoire et d'abonnement](#)

Filtrer par nom... Tous les groupes de re... Tous les types Tous les emplacements Toutes les étiquettes Aucun regrouper

59 éléments ☐ Afficher les types masqués

<input type="checkbox"/>	NOM	TYPE	GRUPE DE RESSOURCES	EMPLACEMENT	ABONNEMENT
<input checked="" type="checkbox"/>	SparkAgileDSS	Azure Databricks Service	admin_spark	Est des États-Unis 2	Visual Studio Enterprise avec MSDN

Launch Notebook

1. Click on **Launch Workspace** tile present on the Cluster Blade.

SparkAgileDSS
Azure Databricks Service

Rechercher (Ctrl+/)

Overview

- Journal d'activité
- Contrôle d'accès (IAM)
- Étiquettes

Paramètres

- Virtual Network Peerings
- Verrous
- Exporter le modèle

Support + dépannage

- Nouvelle demande de support

Details:

- Groupe de ressources (modifier): [admin_spark](#)
- Abonnement (modifier): [Visual Studio Enterprise avec MSDN](#)
- ID d'abonnement: ae6e4f30-ecd0-4f79-86d5-781ac8351e5f
- Managed Resource Group: [databricks-rg-SparkAgileDSS-znmenei4ld6jy](#)
- URL: <https://eastus2.azuredatabricks.net>
- Pricing Tier: standard

Launch Workspace

Documentation **Getting Started** **Import Data from File** **Import Data from Azure Storage**

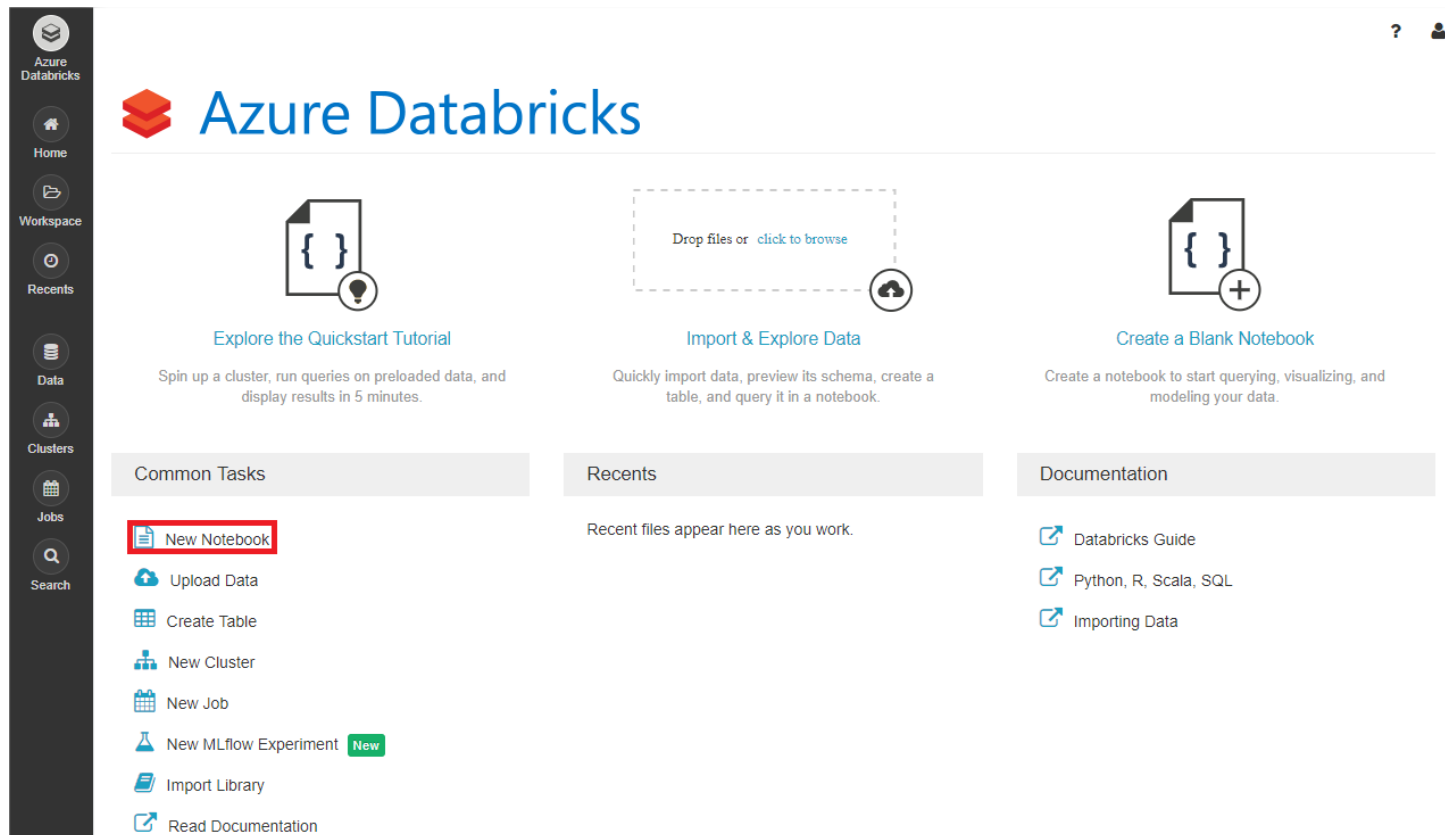
Notebook **Admin Guide**

When prompted, use the admin credential of your Spark Cluster.

Create a new Notebook

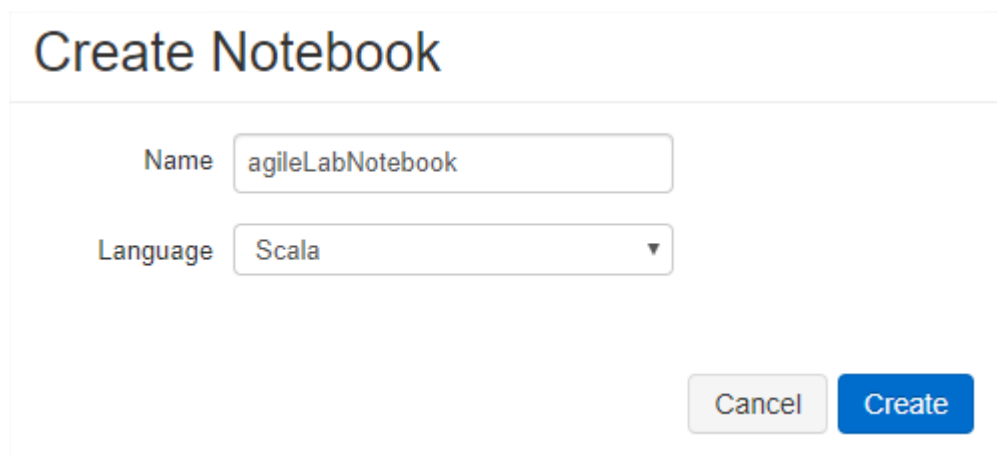
If prompted, enter the admin credentials for the Spark cluster.

Azure Databricks Notebook will open.



The screenshot shows the Azure Databricks home dashboard. On the left is a dark sidebar with navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area features the 'Azure Databricks' logo at the top. Below the logo are three large cards: 'Explore the Quickstart Tutorial', 'Import & Explore Data', and 'Create a Blank Notebook'. At the bottom, there are three sections: 'Common Tasks' (with a 'New Notebook' button highlighted by a red box), 'Recents', and 'Documentation'.

1. Click **New Notebook** button in the middle left or upper right of the screen.
2. Select **Scala** as the language, from the dropdown.
3. Give a name to the note



The 'Create Notebook' dialog box is shown. It has a title 'Create Notebook' at the top. Below the title, there are two input fields: 'Name' with the value 'agileLabNotebook' and 'Language' with a dropdown menu showing 'Scala'. At the bottom right, there are two buttons: 'Cancel' and 'Create'.

Create Spark and SQL context

Starting from Spark2.0 there is no need to import and start SparkContext and SQLContext!

Create data frame from data stored in azure blob storage

A first block of code is already created for you at the top of the screen:

Every time you run a job in Zeppelin, your web browser window title will show a (Busy) status alongside the notebook title.

You will also see a solid circle next to the PySpark text in the top-right corner.

After the job completes, this will change to a hollow circle

1. Paste the following snippet in below empty cell, do not forget to replace `<container_name>` (should be sparklabdata) and `<storage_account_name>`.

```
# Define dataset azure path
Airportspath
="wasb://<container_name>@<storage_account_name>.blob.core.windows.net/Flight/*/*.csv"

# Obtain dataframe
val airports = spark.read.csv(Airportspath)

# show first 20 lines
airports.show()
```

2. Press SHIFT + ENTER. Or Press Play button from tool bar to execute the code inside cell.

```
Cmd 2
1 val airports = spark.read.csv("wasbs://datalake@cbotek.blob.core.windows.net/References/Airports.csv")
2 airports.show()
```

▶ (2) Spark Jobs

▶ airports: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 12 more fields]

	_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	_c10	_c11
1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.081689834590001	145.391998291	5282	10			
2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.20707988739	145.789001465	20	10			

3. Output of above code execution will be as shown below, meaning Spark application correctly started:

Spark Application Id: application_1554229124335_0004

Spark WebUI: http://hn1-sparka.shz1afxuo4we1mdd4ugdg021xg.cx.internal.cloudapp.net:8088/proxy/application_1554229124335_0004/

Took 52 sec. Last updated by anonymous at April 02 2019, 3:11:52 PM.

4. Verify “airports” data type, it should be “DataFrame”. You can paste this code in an empty cell and run it:

```
type(airports)
```

```
<class 'pyspark.sql.dataframe.DataFrame'>
```

```
#try airports alone  
airports
```

DataFrame operations, explore the data

Execute following operations on DataFrame created earlier and observe the output. Use empty cells in the notebook to execute these operations.

1. Do the same thing for another dataset

```
Cmd 5  
1 val root = "wasbs://datalake@cbotek.blob.core.windows.net"  
2 val flightPerf = spark.read.option("header", "true").csv(s"$root}/Flight/*/*")  
3  
4 flightPerf.show()
```

Cancel == Running command...

▶ (1) Spark Jobs

```
flightPerf.count()
```

Output (This will take several minutes):

```
Cmd 6  
1 flightPerf.count()  
  
▶ (1) Spark Jobs  
res9: Long = 161783211
```



2. Sample the data by selecting few years:

```
Cmd 7  
1 import org.apache.spark.sql.functions._  
2 |  
3 val flightPerfSample = flightPerf.filter(col("Year").isin("2017", "2016", "2015", "2014"))  
  
▶ flightPerfSample: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Year: string, Quarter: string ... 108 more fields]  
import org.apache.spark.sql.functions._  
flightPerfSample: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Year: string, Quarter: string ... 108 more fields]
```

3. Look at the data structure:

```
flightPerfSample.printSchema()
```

Output:

Cmd 8

```
1 flightPerfSample.printSchema()
```

```
root
|-- Year: string (nullable = true)
|-- Quarter: string (nullable = true)
|-- Month: string (nullable = true)
|-- DayOfMonth: string (nullable = true)
|-- DayOfWeek: string (nullable = true)
|-- FlightDate: string (nullable = true)
|-- UniqueCarrier: string (nullable = true)
|-- AirlineID: string (nullable = true)
|-- Carrier: string (nullable = true)
|-- TailNum: string (nullable = true)
|-- FlightNum: string (nullable = true)
|-- OriginAirportID: string (nullable = true)
|-- OriginAirportSeqID: string (nullable = true)
|-- OriginCityMarketID: string (nullable = true)
|-- Origin: string (nullable = true)
|-- OriginCityName: string (nullable = true)
|-- OriginState: string (nullable = true)
|-- OriginStateFips: string (nullable = true)
|-- OriginStateName: string (nullable = true)
|-- OriginWac: string (nullable = true)
```

■ ■ ■

We can see that our dataset has quite a lot of columns!

4. Let's display our dataset:

```
flightPerfSample.show()
```

Output... not very readable with our dataset...

Cmd 9

```
1 flightPerfSample.show()
```

- ▶ (1) Spark Jobs

Year Quarter Month DayOfMonth DayOfWeek FlightDate UniqueCarrier AirlineID Carrier TailNum FlightNum OriginAirportID OriginAirportSeqID OriginCityMarketID OriginStateFips OriginStateName OriginWac DestAirportID DestAirportSeqID DestCityMarketID DestCityName DestState DestStateFips DestStateName DepDelay DepDelayMinutes DepDel15 DepartureDelayGroups DepTimeBlk TaxiOut WheelsOff WheelsOn TaxiIn CRSArrTime ArrTime ArrDelay ArrDelayMinutes ArrCancelled CancellationCode Diverted CRSElapsedTime ActualElapsedTime AirTime Flights Distance DistanceGroup CarrierDelay WeatherDelay NASDelay SecurityDelayAddGTime LongestAddGTime DivAirportLandings DivReachedDest DivActualElapsedTime DivArrDelay DivDistance Div1Airport Div1AirportID Div1AirportSeqID Div1TotalGTime Div1WheelsOff Div1TailNum Div2Airport Div2AirportID Div2AirportSeqID Div2WheelsOn Div2TotalGTime Div2LongestGTime Div2WheelsOff Div2TailNum Div3Airport Div3AirportID Div3AirportSeqID Div3WheelsOn Div3TotalGTime Div3LongestGTime Div3WheelsOff Div3TailNum Div4Airport Div4AirportID Div4AirportSeqID Div4WheelsOn Div4TotalGTime Div4WheelsOff Div4TailNum Div5Airport Div5AirportID Div5AirportSeqID Div5WheelsOn Div5TotalGTime Div5LongestGTime Div5WheelsOff Div5TailNum _c109

5. We can select specific columns:

```
flightPerfSample.select("AirlineID", "FlightDate") show()
```

Output:

```
1 flightPerfSample.select("AirlineID","FlightDate").show()
```

▶ (1) Spark Jobs

```
+-----+-----+
|AirlineID|FlightDate|
+-----+-----+
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
| 20355|2014-07-19|
+-----+-----+
```

only showing top 20 rows

6. Apply some filter and show only 1 row:

```
flightPerfSample.select("origin", "dest").filter(col("AirlineID") === 19805).show(1)
```

```
+-----+-----+
|Origin|Dest|
+-----+-----+
|   JFK|  LAX|
+-----+-----+
only showing top 1 row
```

7. We can also rename the output columns:

```
flightPerfSample.select(col("origin").as("FROM"),
col("dest").as("TO")).filter(col("AirlineID") === 19805).show(1)
```

Running SQL Queries

1. To register the DataFrame as SQL table copy below code in empty cell and execute it

```
flightPerfSample.createOrReplaceTempView("flights")
```

2. Then we can work with SQL query using the table we just created

```
spark.sql("show tables").show()
```

```
1 spark.sql("show tables").show()
```

```
+-----+-----+-----+
|database|tableName|isTemporary|
+-----+-----+-----+
|        | flights |         true|
+-----+-----+-----+
```

3. Execute below SQL query and show 10 first lines using the methods we saw above

```
SELECT AirlineId, Year, AirTime, CancellationCode, Cancelled, DepDelay, Dest, Distance, Origin, UniqueCarrier FROM flights
```

Output:

```
Cmd 17
```

```
1 spark.sql("SELECT AirlineId, Year, AirTime, CancellationCode, Cancelled, DepDelay, Dest, Distance, Origin, UniqueCarrier FROM flights").show()
```

▶ (1) Spark Jobs

AirlineId	Year	AirTime	CancellationCode	Cancelled	DepDelay	Dest	Distance	Origin	UniqueCarrier
20355	2014	239.00	null	0.00	11.00	CLT	2125.00	LAX	US
20355	2014	335.00	null	0.00	-8.00	PHX	2979.00	LIH	US
20355	2014	375.00	null	0.00	0.00	LIH	2979.00	PHX	US
20355	2014	340.00	null	0.00	-7.00	PHX	2979.00	LIH	US
20355	2014	319.00	null	0.00	-7.00	SFO	2521.00	PHL	US
20355	2014	131.00	null	0.00	94.00	PHL	1013.00	MIA	US
20355	2014	163.00	null	0.00	1.00	MIA	1013.00	PHL	US
20355	2014	242.00	null	0.00	-1.00	CLT	2125.00	LAX	US
20355	2014	255.00	null	0.00	1.00	LAS	1916.00	CLT	US
20355	2014	217.00	null	0.00	-3.00	CLT	1916.00	LAS	US
20355	2014	270.00	null	0.00	-4.00	LAX	2125.00	CLT	US
20355	2014	136.00	null	0.00	111.00	PHL	1013.00	MIA	US
20355	2014	136.00	null	0.00	18.00	MIA	1013.00	PHL	US
20355	2014	108.00	null	0.00	-12.00	DCA	814.00	TPA	US
20355	2014	130.00	null	0.00	-3.00	MSP	930.00	CLT	US
20355	2014	90.00	null	0.00	-6.00	CLT	590.00	PBI	US
20355	2014	67.00	null	0.00	2.00	PHL	449.00	CLT	US
20355	2014	74.00	null	0.00	32.00	CLT	449.00	PHL	US
20355	2014	300.00	null	0.00	-5.00	PHL	2521.00	SFO	US
20355	2014	315.00	null	0.00	18.00	SFO	2521.00	PHL	US

only showing top 20 rows

4. Let's find out how many rows we have per year:

```
SELECT count(*), Year FROM flights GROUP BY Year ORDER BY Year
```

Output:

```
▶ (1) Spark Jobs
```

Year	count
2014	5819811
2015	5819079
2016	5617658
2017	5210416

5. Verify that the counts are similar here:

```
flightPerfSample.groupBy("YEAR").count().sort("YEAR").show()
```

Notice that 2017 has significantly less flights, and it makes sense because the data is not complete. But what is the last month?

Perform operations on data frames to analyze the data

Use some analytic functions

Some useful functions:

- **groupBy(*cols):** Groups the DataFrame using specified columns, in order to run aggregation on them.
- **count():** Returns the number of rows in DataFrame.
- **collect():** Returns all records as list of row.
- **orderBy(*cols, ascending=True/False):** Returns a new DataFrame sorted by the specified columns.
- **avg(*args):** Computes average values for each numeric column for each group.
- **sum(*args):** Computes sum for each numeric column for each group.

1. Get the number of arrival flights by state in 2014

```
flightPerfSample.filter(col("Year") ===  
2014).groupBy("DestStateName").count().show()
```

Output:

```
+-----+  
| DestStateName | count |  
+-----+  
| Utah          | 112078 |  
| Hawaii        | 96499 |  
| U.S. Virgin Islands | 5123 |  
| Minnesota     | 113085 |  
| U.S. Pacific Trus... | 479 |  
| Ohio          | 82027 |  
| Oregon        | 65458 |  
| Arkansas      | 27904 |  
| Texas         | 717767 |  
| North Dakota  | 15250 |  
| Pennsylvania  | 109574 |  
| Connecticut   | 21780 |  
| Nebraska      | 22837 |  
| Vermont       | 4197 |  
| Nevada        | 153365 |  
| Puerto Rico   | 28114 |  
| Washington    | 121792 |  
| Illinois      | 391833 |  
| Oklahoma      | 40491 |  
| Delaware      | 711 |  
+-----+  
only showing top 20 rows
```

2. **Try by yourself:** Select top 5 States from previous output

3. **Try by yourself:** For those 5 states, calculate the number of flights variation (in %), year over year (from 2014 to 2015, and 2015 to 2016).

Here is the desired output:

```

▶ (6) Spark Jobs
▶ res: org.apache.spark.sql.DataFrame = [DestStateName: string, 2014: long ... 6 more fields]
+-----+-----+-----+-----+-----+-----+-----+
|DestStateName| 2014| 2015| 2016| 2017| 2014_to_2015| 2015_to_2016| 2016_to_2017|
+-----+-----+-----+-----+-----+-----+-----+
|      Texas|717767|688031|578440|511105|-4.142848584568526|-15.928206723243576|-11.64079247631561|
|   Illinois|391833|418264|340426|334513| 6.745475751149073|-18.609777556758416|-1.7369413617056182|
|    Georgia|386052|394412|398518|348187| 2.1655113818863896| 1.0410433759622038|-12.629542454795015|
|    Florida|428056|449248|447168|416851| 4.950754106939286|-0.4629959398817647|-6.779778517246314|
| California|739170|707762|727407|692216|-4.249090195760104| 2.775650571802373|-4.8378693083789415|
+-----+-----+-----+-----+-----+-----+-----+

res: org.apache.spark.sql.DataFrame = [DestStateName: string, 2014: bigint ... 6 more fields]

```

There are multiple ways of achieving this, for example:

- We could filter the dataset in order to have only the states we found in the last query
- Next we can group the data per **DestinationStateName** and pivot per **Year**
- Then we can count the number of rows
- And finally compute the difference between 2014 and 2015, 2015 and 2016

Bonus: Can you try to do this with a window function ?


Learn how to JOIN dataset

1. Load another dataset containing the Cancellation References

Cmd 23

```
1 val root = "wasbs://datalake@cbotek.blob.core.windows.net"
2 val refAnnulations = spark.read.option("header", "true").csv(s"$root/References/RefAnnulations.csv")
3 refAnnulations.show()
```

▶ (2) Spark Jobs

▶  refAnnulations: org.apache.spark.sql.DataFrame = [Code: string, Description: string]

```
+-----+
|Code|      Description|
+-----+
| A |           Carrier|
| B |           Weather|
| C |National Air System|
| D |           Security|
+-----+
```

```
root: String = wasbs://datalake@cbotek.blob.core.windows.net
refAnnulations: org.apache.spark.sql.DataFrame = [Code: string, Description: string]
```

2. Show top 5 origin cities having the most flight cancellation

```
flightPerfSample
  .filter(col("Cancelled") === 1)
  .groupBy("OriginCityName", "CancellationCode")
  .count()
  .orderBy(desc("count"))
  .join(refAnnulations, col("CancellationCode") === col("Code")).show()
```

Output:

```
+-----+-----+-----+-----+
|OriginCityName|CancellationCode|count|Code|      Description|
+-----+-----+-----+-----+
|Chicago, IL|B|20367|B|           Weather|
|Dallas/Fort Worth...|B|11357|B|           Weather|
|New York, NY|B|10672|B|           Weather|
|Chicago, IL|C| 9822|C|National Air System|
|Atlanta, GA|B| 9703|B|           Weather|
|Houston, TX|B| 9401|B|           Weather|
|Newark, NJ|C| 7153|C|National Air System|
|Chicago, IL|A| 6534|A|           Carrier|
|Denver, CO|B| 6528|B|           Weather|
|New York, NY|C| 6228|C|National Air System|
|Dallas/Fort Worth...|A| 5500|A|           Carrier|
|San Francisco, CA|B| 5421|B|           Weather|
|Boston, MA|B| 5404|B|           Weather|
|Washington, DC|B| 5375|B|           Weather|
|New York, NY|A| 5303|A|           Carrier|
|Newark, NJ|B| 4995|B|           Weather|
|Los Angeles, CA|A| 4834|A|           Carrier|
|Baltimore, MD|B| 4215|B|           Weather|
|Atlanta, GA|A| 4112|A|           Carrier|
|Orlando, FL|B| 4093|B|           Weather|
+-----+-----+-----+-----+
```

Data type conversion and statistical functions

One of the main advantage of PySpark/Scala over SQL is the access to a ton of libraries, for statistical purpose and matrix calculation for example.

1. As a simple example, calculate the correlation coefficient between the AIR_TIME and DISTANCE. For that we can use the function "corr", taking in arguments 2 columns of a dataframe (using the Pearson method).

- Let's try this:

```
flightPerfSample.stat.corr("AirTime", "Distance")
```

Output:

```
java.lang.IllegalArgumentException: requirement failed: Currently correlation calculation for columns with dataType string not supported.
    at scala.Predef$.require(Predef.scala:224)
    at org.apache.spark.sql.execution.stat.StatFunctions$$anonfun$collectStatisticalData$3$.apply(StatFunctions.scala:159)
    at org.apache.spark.sql.execution.stat.StatFunctions$$anonfun$collectStatisticalData$3$.apply(StatFunctions.scala:157)
    at scala.collection.immutable.List.foreach(List.scala:392)
    at org.apache.spark.sql.execution.stat.StatFunctions$.collectStatisticalData(StatFunctions.scala:157)
    at org.apache.spark.sql.execution.stat.StatFunctions$.pearsonCorrelation(StatFunctions.scala:169)
    at org.apache.spark.sql.DataFrameStatFunctions.corr(DataFrameStatFunctions.scala:160)
    at org.apache.spark.sql.DataFrameStatFunctions.corr(DataFrameStatFunctions.scala:180)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:61)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:61)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:63)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:65)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:67)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:69)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:71)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:73)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:75)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:77)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:79)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:81)
    at line5af61aaa5dba49f68051686d69538d60134.$read$$iw$$iw$$iw$$iw$$iw$$iw$.<init>(command-1987597012917638:83)
```

Oops... corr function is based on numeric values, and it looks like Spark is not automatically converting our strings into numeric values.

- Manually cast the data and assign result into a new dataframe:

```
import org.apache.spark.sql.types._
val newFlightPerfSample = flightPerfSample.select(col("AirTime").cast(FloatType),
  $"Distance" cast "float")
```

- Try again the correlation calculation

```
newFlightPerfSample.stat.corr("AIR_TIME", "DISTANCE")
```

Output:

```
import org.apache.spark.sql.types._
newFlightPerfSample: org.apache.spark.sql.DataFrame = [AirTime: float, Distance: float]
res51: Double = 0.9615025690006345
```

Nearly perfect correlation (coefficient is always between -1 and 1), but you already probably guessed it, as this correlation is quite obvious...

Visualize the results

Try by yourself: Find out the State destination with the bigger difference in 2016, in term of number of flights, between 2 months (variation in %).

- To resolve this, first you can build a temp table containing the count of flights by MONTH / DEST_STATE_NAME
- From here you can calculate the variation in % with a window function. Here is how we create a window. We will use this to compute the variation

```
val windowSpec = Window.partitionBy("DestStateName").orderBy("Month")
```

Output:

```
+-----+-----+-----+-----+
|DestStateName|Month|count|month_to_month|
+-----+-----+-----+-----+
|California   |3    |59449|6504          |
|California   |10   |62267|6042          |
|Florida      |3    |43998|5702          |
|Texas        |3    |49840|4644          |
|Georgia      |3    |34146|4382          |
|Florida      |12   |40383|4201          |
|Illinois     |10   |29801|3817          |
|Illinois     |3    |27950|3428          |
|Georgia      |10   |34004|3092          |
|Florida      |11   |36182|2920          |
|California   |5    |61471|2889          |
|Colorado     |3    |21613|2874          |
|Arizona      |3    |16280|2348          |
|New York     |3    |22043|2211          |
|California   |7    |65622|2086          |
|California   |6    |63536|2065          |
|Illinois     |5    |29954|2033          |
|California   |12   |61339|1934          |
|Michigan     |10   |13477|1765          |
|Texas        |5    |49070|1706          |
+-----+-----+-----+-----+
only showing top 20 rows
```

Spoiler

```
val import org.apache.spark.sql.expressions._
import org.apache.spark.sql.functions._

val countPerMonth = flightPerf.filter(col("YEAR") ===
2016).groupBy("DestStateName", "Month").count()

val windowSpec = Window.partitionBy("DestStateName").orderBy("Month")

val monthToMonthDiff =
countPerMonth
```

```
.withColumn("month_to_month", $"count" - lag($"count", 1).over(windowSpec))  
.filter($"month_to_month".isNotNull)  
.orderBy($"month_to_month".desc)  
  
monthToMonthDiff.show(false)
```

1. Our winner should be California. Let's visualize the month trend for this state (here we assume that a temp table "flightMonthTable" has been created, containing the count of flights by MONTH, YEAR and DEST_STATE_NM):

```
countPerMonth.filter($"DestStateName" === "California").orderBy($"Month").show(12)
```

Output:

```
-----+-----+-----+
|DestStateName|Month|count|
-----+-----+-----+
|California|1|56225|
|California|10|62267|
|California|11|59405|
|California|12|61339|
|California|2|52945|
|California|3|59449|
|California|4|58582|
|California|5|61471|
|California|6|63536|
|California|7|65622|
|California|8|66303|
|California|9|60263|
-----+-----+-----+
```

Section 3 - Power BI on Spark With Databricks

To Design a Power BI report based on Spark, we need to persist our data into a Hive table.

Dataframe to HIVE

1. Create a new notebook
2. Create the hive table with this data: the number of flights and average delay (DEP_DELAY) by destination state for each departure city. To be more representative we will only consider the flights having a delay > 1 hour.
 - Re-create the flight sample dataframe from the previous part

```
import org.apache.spark.sql.functions._

spark.conf.set(
  "fs.azure.account.key.cbotek.blob.core.windows.net",
  "0tCbVawj0BniiLxMgJfeq878iWV8MqUYp3klz76+67wvtUOKDShSRS4MCclv/PYQqrZNNxcj+17sk6BUBd
kcYA==")
val root = "wasbs://datalake@cbotek.blob.core.windows.net"

val refAnnulations = spark.read.option("header",
"true").csv(s"${root}/References/RefAnnulations.csv")

val airports = spark.read.csv(s"${root}/References/Airports.csv")

val routes = spark.read.option("header",
"false").csv(s"${root}/References/Routes.csv")

val data2014 = spark.read.option("header", "true").csv(s"${root}/Flight/2014/*")
val data2015 = spark.read.option("header", "true").csv(s"${root}/Flight/2015/*")
val data2016 = spark.read.option("header", "true").csv(s"${root}/Flight/2016/*")
val data2017 = spark.read.option("header", "true").csv(s"${root}/Flight/2017/*")
val flightPerfSample = data2014.union(data2015).union(data2016).union(data2017)
```

- Build our query and assign it to a new dataframe

```
# Register a temp table
flightPerfSample.registerTempTable("departureDelays")

# New dataframe
val AvgDelay =
spark
.sql("SELECT  OriginCityName, DestStateName, 'United States' as Country,
AVG(DepDelay) as AverageDelay, COUNT(*) as DelayFrequency FROM departureDelays
WHERE DepDelay > 60 GROUP BY OriginCityName, DestStateName")
AvgDelay.createOrReplaceTempView("avgDelay")
```

- You can check if the table was created successfully by calling 'show tables'

```
spark.sql("show tables").show()
```

- At this point our analysis table is temporary

Cmd 4

```
1 spark.sql("SHOW TABLES").show()
```

```
+-----+-----+-----+
|database|   tableName|isTemporary|
+-----+-----+-----+
|        |      avgdelay|         true|
|        |departuredelays|         true|
+-----+-----+-----+
```

- In order to create a table in Hive we need to execute the line below:

```
spark.sql("create table DestinationStateAverageDelayAnalysis as select * from
avgDelay")
```


3. Let's see what's the difference now when we execute show tables again

```
spark.sql("SHOW TABLES").show()
```

Please compare your result and make sure you all have something similar

```
+-----+-----+-----+
|database|tableName|isTemporary|
+-----+-----+-----+
|default|destinationstateaveragedelayanalysis|false|
|        |avgdelay|true|
|        |departuredelays|true|
+-----+-----+-----+
```

4. At this point if we go back to our blob storage we do not see any differences. So where does this table was save exactly?

Home > cbotek - Blobs > datalake

datalake
Container

Search (Ctrl+/) «

Upload Refresh Change access level Delete Acquire lease Break lease View snapshots Create snapshot

Authentication method: Access key (Switch to Azure AD User Account)
Location: datalake

Search blobs by prefix (case-sensitive) ☐ Show deleted blobs

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
Flight					-
References					-

5. Databricks is storing its meta data on a different blob storage which we cannot access

6. If we go back to the azure portal you should see a blob storage with a name similar to mine:

Home > Storage accounts

Storage accounts
agiledss (Default Directory)

+ Add Edit columns Refresh Assign tags Delete

Subscriptions: Pay-As-You-Go

Filter by name... All resource groups All types All locations All tags No gr...

2 items

NAME	TYPE	KIND	RESOURCE GROUP	LOCATION	SUBSCRIPTION
cbotek	Storage account	Storage	cbotek	East US 2	Pay-As-You-Go
dbstorage3o4zce4odpuoe	Storage account	BlobStorage	databricks-rg-labSparkAgileds...	East US	Pay-As-You-Go

If we click on it we can look at the details

Home > dbstorage3o4zce4odpuoe

dbstorage3o4zce4odpuoe
Storage account

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Data transfer

Events

Storage Explorer (preview)

Settings

Access keys

Geo-replication

CORS

Open in Explorer → Move Delete Refresh

Resource group (change) : databricks-rg-labSparkAgiledss-kshnmhiaqovoo

Status : Primary: Available, Secondary: Available

Location : East US, West US

Subscription (change) : Pay-As-You-Go

Subscription ID : 66886784-7e19-470c-b315-f25f857c2929

Tags (change) : application : databricks databricks-environment : true

Performance/Access tier : Standard/Hot

Replication : Geo-redundant storage (GRS)

Account kind : BlobStorage

Services

Blobs
REST-based object storage for unstructured data
[Learn more](#)

But we cannot access the files

Home > Storage accounts > dbstorage3o4zce4odpuoe - Storage Explorer (preview)

Storage accounts
agiledss (Default Directory)

+ Add Edit columns More

Filter by name...

NAME

cbotek

dbstorage3o4zce4odpuoe

dbstorage3o4zce4odpuoe - Storage Explorer (preview)
Storage account

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Data transfer

Events

Storage Explorer (preview)

Access blocked

The resource is locked

Cannot access the data plane because of a read lock on the resource or its parent.

7. Now go back to the notebook tab, and type the following command to query your table

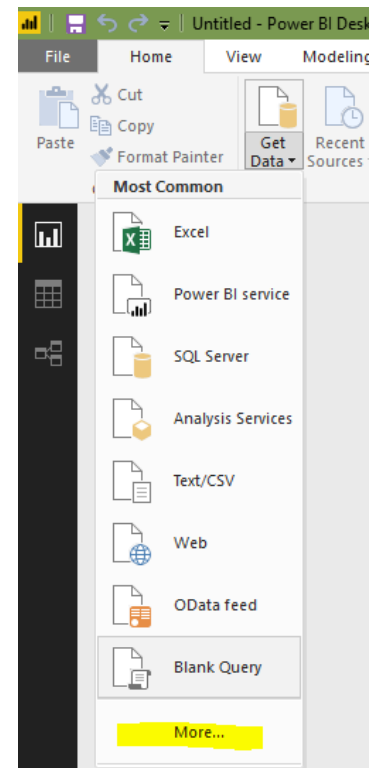
```
spark.sql("Select * from destinationStateAverageDelayAnalysis Limit 5").show(false)
```

```
+-----+-----+-----+-----+-----+
|OriginCityName|DestStateName|Country|AverageDelay|DelayFrequency|
+-----+-----+-----+-----+-----+
|Sacramento, CA|North Carolina|United States|112.52272727272727|44|
|Chicago, IL|Massachusetts|United States|118.01713673687969|2801|
|Baltimore, MD|New York|United States|112.8171466845278|1493|
|Tampa, FL|Indiana|United States|118.55932203389831|236|
|Pittsburgh, PA|Tennessee|United States|105.38461538461539|39|
+-----+-----+-----+-----+-----+
```

Connect an Azure Databricks Spark Datasource

In this exercise, you'll connect Power BI to the previous hive table.

1. Open your Microsoft Power BI Desktop application
2. With a new report, inside the **Home** tab, expand the **Get Datasource** menu and select the **More...** option
3. In the **Get Data** dialog window, on the left side, select **Spark**.



Get Data

spark

All

Azure

Online Services

Other

All

☆ Azure HDInsight Spark

🔥 SparkPost (Beta)

☆ Spark

Certified Connectors

Connect

Cancel

4. Click **“Connect”**

×

Spark

Server ⓘ

Example: `http://example.com:10000/cliservice`

Protocol

> Advanced Options (optional)

Data Connectivity mode ⓘ

☒ Import

☐ DirectQuery

OK Cancel

5. In order to find these informations, let's go back to Databricks and click on the left hand side on **Clusters**

Azure Databricks

Home

Workspace

Recents

Data

Clusters

Clusters

+ Create Cluster

▼ Interactive Clusters

Name	State
aaa	Running ⓘ

▼ Job Clusters

6. Then click on your cluster

The screenshot shows the Azure Databricks Clusters management interface. On the left is a sidebar with navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main content area is titled 'Clusters / aaa' and includes buttons for Edit, Clone, Restart, Terminate, and Delete. Below these are tabs for Configuration, Notebooks (1), Libraries, Event Log, Spark UI, Driver Logs, Apps, and Spark Cluster L. The 'Configuration' tab is active, showing 'Worker Type' as Standard_DS3_v2 with 14.0 GB Memory, 4 Cores, and 0.75 DBU, and 'Min Workers' set to 2 and 'Max Workers' set to 8. The 'Driver Type' is also Standard_DS3_v2 with the same specifications. Under 'Advanced Options', the 'JDBC/ODBC' tab is selected. It displays the 'Server Hostname' as eastus.azuredatabricks.net, 'Port' as 443, and 'Protocol' as HTTPS. The 'HTTP Path' section shows two options: a unique path for the current cluster and an alias path for cluster 'aaa'. The 'JDBC URL' section provides two example URLs, with the second one (using the alias) being highlighted. A link to 'Learn more about connecting your favorite BI tool to Databricks' is at the bottom.

Extract the base url: eastus.azuredatabricks.net:443 and add your unique HTTP Path:
sql/protocolv1/o/3641349854446370/0614-201836-gaze968

Here is the final url to put in Power BI:

https://eastus.azuredatabricks.net:443/sql/protocolv1/o/3641349854446370/0614-201836-gaze968

7. Copy the url and paste it in Power BI, then click on connect

Windows

Username / Password

Spark

✕

☆ https://eastus.azuredatabricks.net:443/sql/protocol...

User name

Password

Back

Connect

Cancel

Make sure you see this page or ask for help 😊

- Now let's resolve the username/password in order to connect to our cluster. Go back to Databricks and click on the top right corner, **User settings**

PORTAL botek.christophe@agiledss.com

? 👤

Signed in as
botek.christophe@agile...

User Settings

Admin Console

Manage Account

Log Out

Workspaces

✓ labSparkAgiledss
botek.christophe@agiled...

You should see this

9. Now click on **Generate New Token**, then ok, and then copy the token
10. Now we can go back to Power BI, paste the token in the password field and enter 'token' as username. Like so:

11. Click **Connect**.
12. In the Navigator dialog window, expand the HIVE database, and then expand **<your_cluster_name>.azuredatabricks.net**
13. Make sure you see the Hive table we created earlier.

Navigator

The Navigator window displays a table titled "destinationstateaveragedelayanalysis". The table has four columns: OriginCityName, DestStateName, Country, and AverageDelay. The data is as follows:

OriginCityName	DestStateName	Country	AverageDelay
Sacramento, CA	North Carolina	United States	112.52
Chicago, IL	Massachusetts	United States	118.01
Baltimore, MD	New York	United States	112.81
Tampa, FL	Indiana	United States	118.5
Pittsburgh, PA	Tennessee	United States	105.38
Tulsa, OK	Nevada	United States	135.0
Ontario, CA	Colorado	United States	121.18
Jacksonville, FL	Massachusetts	United States	121.31
Atlanta, GA	Wisconsin	United States	124.9
Nashville, TN	Georgia	United States	143.91
Knoxville, TN	Georgia	United States	143.12
Santa Ana, CA	Georgia	United States	165.67
Los Angeles, CA	Minnesota	United States	123.17
Des Moines, IA	New Jersey	United States	18

14. Click **Load**.

15. Explore your data model in the diagram tab at the left.

The data will be loaded into the Power BI Desktop file.

Once loaded, in the **Queries** pane (located at the left), select the query to review the data from the Hive table.

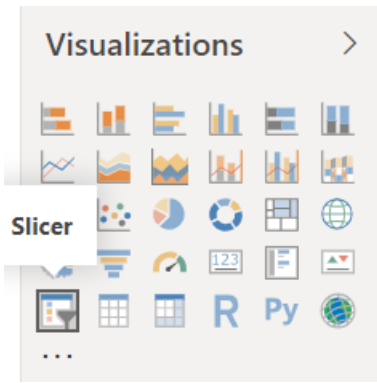
The Queries pane shows a list of queries. The query "destinationstateaver..." is selected, and its fields are displayed in a list:

- AverageDelay
- Country
- DelayFrequency
- DestStateName
- OriginCityName

Designing the Power BI report

In this exercise, you will design an interactive report based on the hive table.

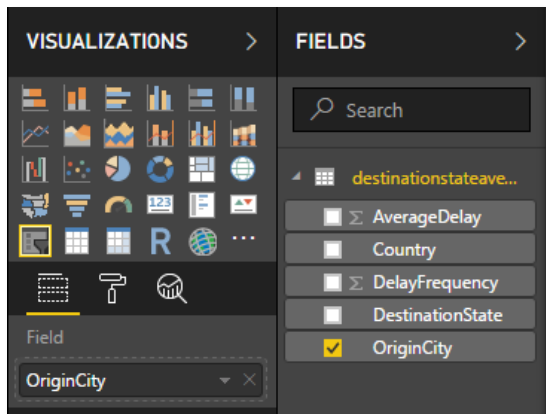
1. Go to the report pane
2. To add a Segment from inside the Visualization pane, click the **Slicer** icon



3. Reposition and resize the visualization based on the following diagram.

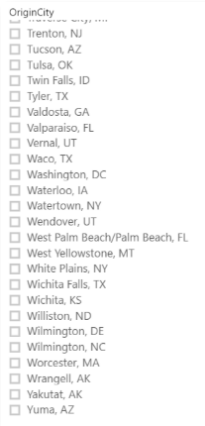


4. In the Fields pane (located at the right), Expand the **destinationStateAverageDelayAnalysis** table.



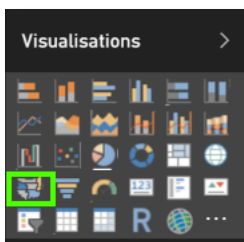
5. From the Fields pane, inside the expanded table, check the **OriginCity** field.

Verify that the visualization looks like the following

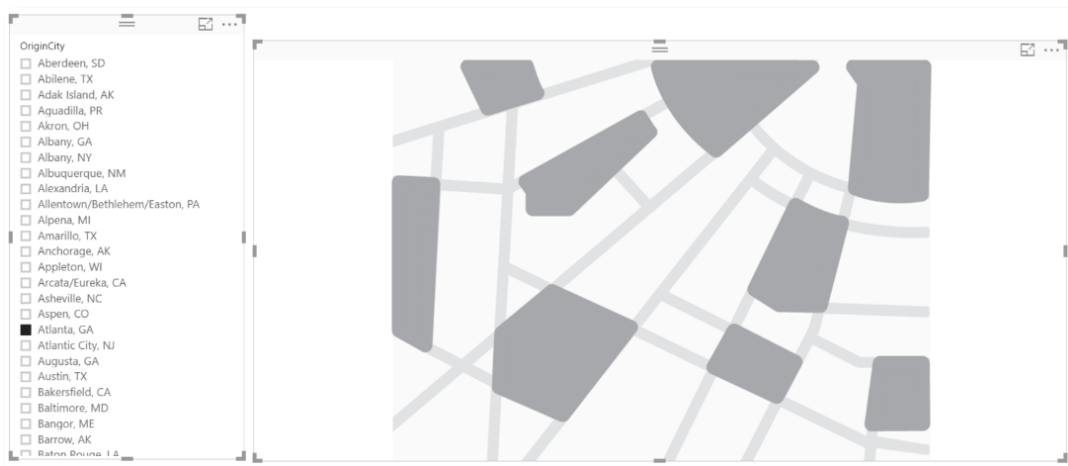


6. To add a Map, from inside the Visualization pane, click on the **Filled Map** icon.

Tips: you can hover the cursor over each icon to reveal a tooltip describing the type of visualization.

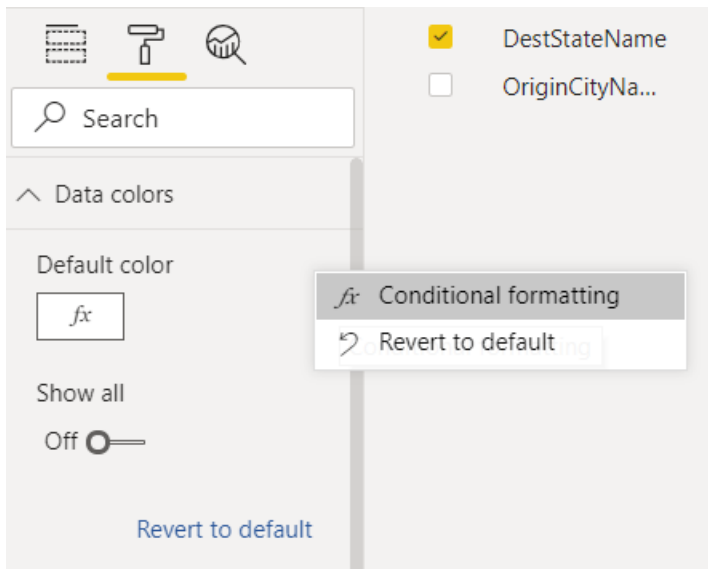


7. Reposition and resize the map visualization based on the following diagram.

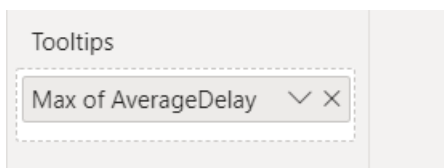


8. From the Fields pane, inside the expanded table, drag the **DestinationState** to Emplacement property and repeat the operation with the **Country** below the **DestinationState**.

9. From the Format pane, click on Data Colors and then on Conditional Formatting. Choose the range of color you like the most to represent the minimum and the maximum values.



10. From the Fields pane, from inside the expanded table, drag the **AverageDelay**, to the Tool Tips property. You can play a little with the different calculation offered. I selected the Maximum of Average delay:

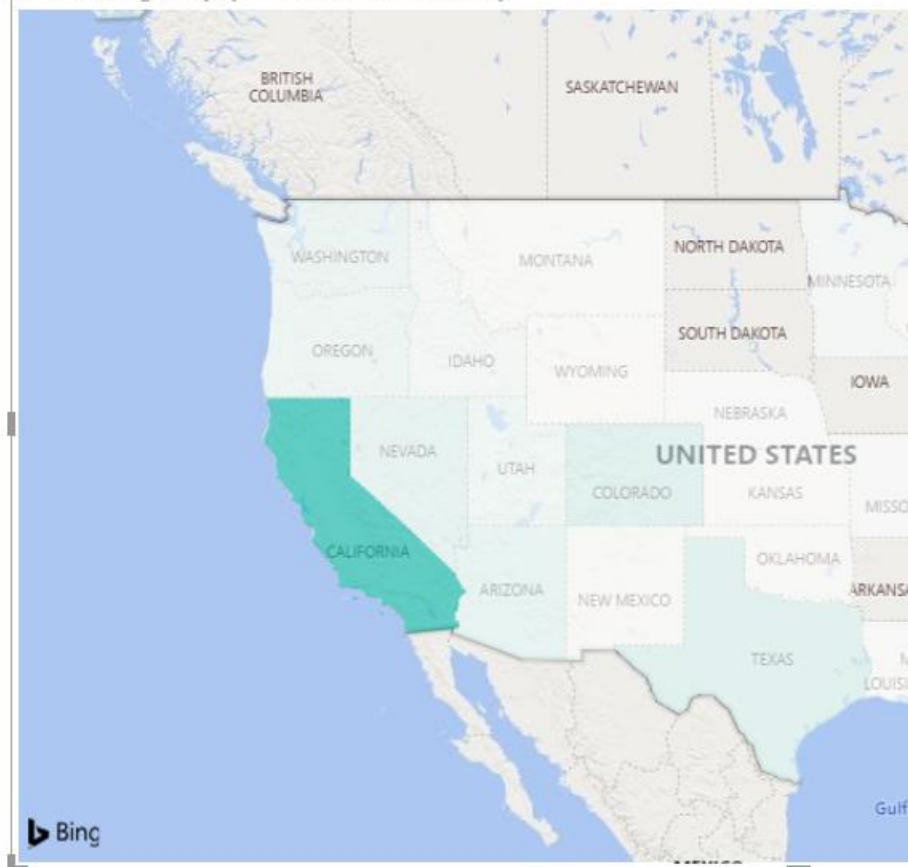


11. Verify that the visualization looks like the following

OriginCityName

- ☐ Santa Rosa, CA
- ☐ Sarasota/Bradenton, FL
- ☐ Sault Ste. Marie, MI
- ☐ Savannah, GA
- ☐ Scranton/Wilkes-Barre, PA
- ☒ Seattle, WA
- ☐ Shreveport, LA
- ☐ Sioux City, IA
- ☐ Sioux Falls, SD
- ☐ Sitka, AK
- ☐ South Bend, IN
- ☐ Spokane, WA
- ☐ Springfield, IL
- ☐ Springfield, MO
- ☐ St. Augustine, FL
- ☐ St. Cloud, MN
- ☐ St. George, UT
- ☐ St. Louis, MO
- ☐ State College, PA
- ☐ Sun Valley/Hailey/Ketchum, ID
- ☐ Syracuse, NY
- ☐ Tallahassee, FL
- ☐ Tampa, FL
- ☐ Texarkana, AR
- ☐ Toledo, OH
- ☐ Topeka, KS

Max of AverageDelay by DestStateName and Country



User Define Function

In this final exercise you will create a new Hive table, and connect a Power BI visualization on it to display the traffic flow. We will use this exercise to introduce to you the RDD API and the user define functions.

1. Go back to the previous notebook and paste the following in order to create a path to the airports CSV file

```
val airportsPath = s"${root}/References/Airports.csv"
```

2. Instantiate a dataframe as a textfile this time

```
val airportsDf = spark.read.text(airportsPath)
```

```
airportsDf.show(false)
```

Output:

```

+-----+
|value
+-----+
|1,"Goroka Airport","Goroka","Papua New Guinea","GKA","AYGA",-6.081689834590001,145.391998291,5282,10,"U","Pacific/Port_Moresby","airport",
|2,"Madang Airport","Madang","Papua New Guinea","MAG","AYMD",-5.20707988739,145.789001465,20,10,"U","Pacific/Port_Moresby","airport",
|3,"Mount Hagen Kagamuga Airport","Mount Hagen","Papua New Guinea","HGU","AYMH",-5.826789855957031,144.29600524902344,5388,10,"U","Pac
|4,"Nadzab Airport","Nadzab","Papua New Guinea","LAE","AYNZ",-6.569803,146.725977,239,10,"U","Pacific/Port_Moresby","airport","OurAirp
|5,"Port Moresby Jacksons International Airport","Port Moresby","Papua New Guinea","POM","AYPY",-9.443380355834961,147.22000122070312,
|6,"Wewak International Airport","Wewak","Papua New Guinea","WwK","AYWK",-3.58383011818,143.669006348,19,10,"U","Pacific/Port_Moresby"
|7,"Narsarsuaq Airport","Narssarssuaq","Greenland","UAK","BGBW",61.1604995728,-45.4259986877,112,-3,"E","America/Godthab","airport",
|8,"Godthaab / Nuuk Airport","Godthaab","Greenland","GOH","BGGH",64.19090271,-51.6781005859,283,-3,"E","America/Godthab","airport",
|9,"Kangerlussuaq Airport","Sondrestrom","Greenland","SFJ","BGSF",67.0122218992,-50.7116031647,165,-3,"E","America/Godthab","airport",
|10,"Thule Air Base","Thule","Greenland","THU","BGTl",76.5311965942,-68.7032012939,251,-4,"E","America/Thule","airport","OurAirports"
|11,"Akureyri Airport","Akureyri","Iceland","AEY","BIAR",65.66000366210938,-18.07270050048828,6,0,"N","Atlantic/Reykjavik","airport",
|12,"Egilsstaðir Airport","Egilsstaðir","Iceland","EGS","BIEG",65.2833023071289,-14.401399612426758,76,0,"N","Atlantic/Reykjavik","air

```

3. We will need to create a User Defined function in order to split each line into an array, trim the data and remove the double quotes

```

import org.apache.spark.sql.types._

import org.apache.spark.sql.functions._

val clean: String => Array[String] = _.split(",").map(_.replace("\"", "").trim())

val cleanUDF = udf(clean)

val airportsDfCleaned = airportsDf.withColumn("value", cleanUDF(col("value")))

airportsDfCleaned.show(false)

```

This is a bit better but we will need to get each value into a separated column.

```

+-----+
|value
+-----+
|[1, Goroka Airport, Goroka, Papua New Guinea, GKA, AYGA, -6.081689834590001, 145.391998291, 5282, 10, U, Pacific/Port_Moresby, ai
|[2, Madang Airport, Madang, Papua New Guinea, MAG, AYMD, -5.20707988739, 145.789001465, 20, 10, U, Pacific/Port_Moresby, airport,
|[3, Mount Hagen Kagamuga Airport, Mount Hagen, Papua New Guinea, HGU, AYMH, -5.826789855957031, 144.29600524902344, 5388, 10, U,
|[4, Nadzab Airport, Nadzab, Papua New Guinea, LAE, AYNZ, -6.569803, 146.725977, 239, 10, U, Pacific/Port_Moresby, airport, OurAir
|[5, Port Moresby Jacksons International Airport, Port Moresby, Papua New Guinea, POM, AYPY, -9.443380355834961, 147.2200012207031
|[6, Wewak International Airport, Wewak, Papua New Guinea, WwK, AYWK, -3.58383011818, 143.669006348, 19, 10, U, Pacific/Port_Mores
|[7, Narsarsuaq Airport, Narssarssuaq, Greenland, UAK, BGBW, 61.1604995728, -45.4259986877, 112, -3, E, America/Godthab, airport,
|[8, Godthaab / Nuuk Airport, Godthaab, Greenland, GOH, BGGH, 64.19090271, -51.6781005859, 283, -3, E, America/Godthab, airport, 0
|[9, Kangerlussuaq Airport, Sondrestrom, Greenland, SFJ, BGSF, 67.0122218992, -50.7116031647, 165, -3, E, America/Godthab, airport
|[10, Thule Air Base, Thule, Greenland, THU, BGTl, 76.5311965942, -68.7032012939, 251, -4, E, America/Thule, airport, OurAirports]
|[11, Akureyri Airport, Akureyri, Iceland, AEY, BIAR, 65.66000366210938, -18.07270050048828, 6, 0, N, Atlantic/Reykjavik, airport,
|[12, Egilsstaðir Airport, Egilsstaðir, Iceland, EGS, BIEG, 65.2833023071289, -14.401399612426758, 76, 0, N, Atlantic/Reykjavik, a

```

4. In order to do this, we can figure out the number of values in each line and ask spark to create a column for each index:

```

//In our case we counted 15 different values for each line

val airportsDfSplitted = airportsDfCleaned.select((0 until 14).map(i => col("value")(i).alias(s"col_$i")): _*)

airportsDfSplitted.show()

```

Output:

► (1) Spark Jobs

►  airportsDfSplitted: org.apache.spark.sql.DataFrame = [col_0: string, col_1: string ... 12 more fields]

col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.081689834590001	145.391998291	5282	10	U P
2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.20707988739	145.789001465	20	10	U P
3	Mount Hagen Kagam...	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.826789855957031	144.29600524902344	5388	10	U P
4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	AYNZ	-6.569803	146.725977	239	10	U P
5	Port Moresby Jack...	Port Moresby	Papua New Guinea	POM	AYPY	-9.443380355834961	147.22000122070312	146	10	U P
6	Wewak Internation...	Wewak	Papua New Guinea	WWK	AYWK	-3.58383011818	143.669006348	19	10	U P
7	Narsarsuaq Airport	Narssarssuaq	Greenland	UAK	BGBW	61.1604995728	-45.4259986877	112	-3	E
8	Godthaab / Nuuk A...	Godthaab	Greenland	GOH	BGGH	64.19090271	-51.6781005859	283	-3	E
9	Kangerlussuaq Air...	Sondrestrom	Greenland	SFJ	BGSF	67.0122218992	-50.7116031647	165	-3	E
10	Thule Air Base	Thule	Greenland	THU	BGTL	76.5311965942	-68.7032012939	251	-4	E
11	Akureyri Airport	Akureyri	Iceland	AEY	BIAR	65.66000366210938	-18.07270050048828	6	0	N
12	Reykjavik Airport	Reykjavik	Iceland	EGS	RTFG	65.2833023071289	-14.401399612426758	76	0	N

5. Last thing we need to do is to apply a schema to this dataframe

```
val schema =
```

```
StructType(List(
  StructField("AirportId", StringType, true),
  StructField("Name", StringType, true),
  StructField("City", StringType, true),
  StructField("Country", StringType, true),
  StructField("IATA", StringType, true),
  StructField("ICAO", StringType, true),
  StructField("Latitude", StringType, true),
  StructField("Longitude", StringType, true),
  StructField("Altitude", StringType, true),
  StructField("Timezone", StringType, true),
  StructField("DST", StringType, true),
  StructField("TzDatabase", StringType, true),
  StructField("Type", StringType, true),
  StructField("Source", StringType, true)))
```

```
val airportsWithSchema = airportsDfSplitted.sqlContext.createDataFrame(airportsDfSplitted.rdd, schema)
airportsWithSchema.show(false)
```

6. Create temporary view based on the two DataFrames

```
// Creates a temporary view based on the DataFrame
airportsWithSchema.createOrReplaceTempView("airports_na")
flightPerfSample.createOrReplaceTempView("departureDelays")
```

7. Do the projection of Flights with the enrichment of the Latitude and Longitude of each Airport's location.

```
// We need to rename the columns and select the ones interesting to our analysis
val flights = flightPerfSample
  .select(
    $"OriginStateName".as("origin_state"),
    $"OriginCityName".as("origin_city"),
    $"Origin".as("origin_airport"),
    $"DestStateName".as("destination_state"),
    $"DestCityName".as("destination_city"),
    $"Dest".as("destination_airport"),
    $"DepDelay".as("dep_delay"))

//We also need to cast longitude and latitude as double
val airports = airportsWithSchema
  .withColumn("Latitude", $"Latitude".cast(DoubleType))
  .withColumn("Longitude", $"Longitude".cast(DoubleType))

//then we can proceed with the aggregation
val airport_traffic = flights
  .groupBy("origin_state", "origin_city", "origin_airport", "destination_state", "destination_city", "destination_airport")
  .agg(count("*").as("FlightCount"), avg("dep_delay").as("dep_delay"))
  .join(airports.select($"IATA", $"Latitude".as("origin_latitude"), $"Longitude".as("origin_longitude")), $"origin_airport" === $"IATA", "left")
  .drop("IATA")
  .join(airports.select($"IATA", $"Latitude".as("des_latitude"), $"Longitude".as("des_longitude")), $"destination_airport" === $"IATA", "left")

//and finally we can save the result as a non temporary table
airport_traffic.write.saveAsTable("airports_traffic")

//check if the table was saved correctly
spark.sql("show tables").show(false)
```

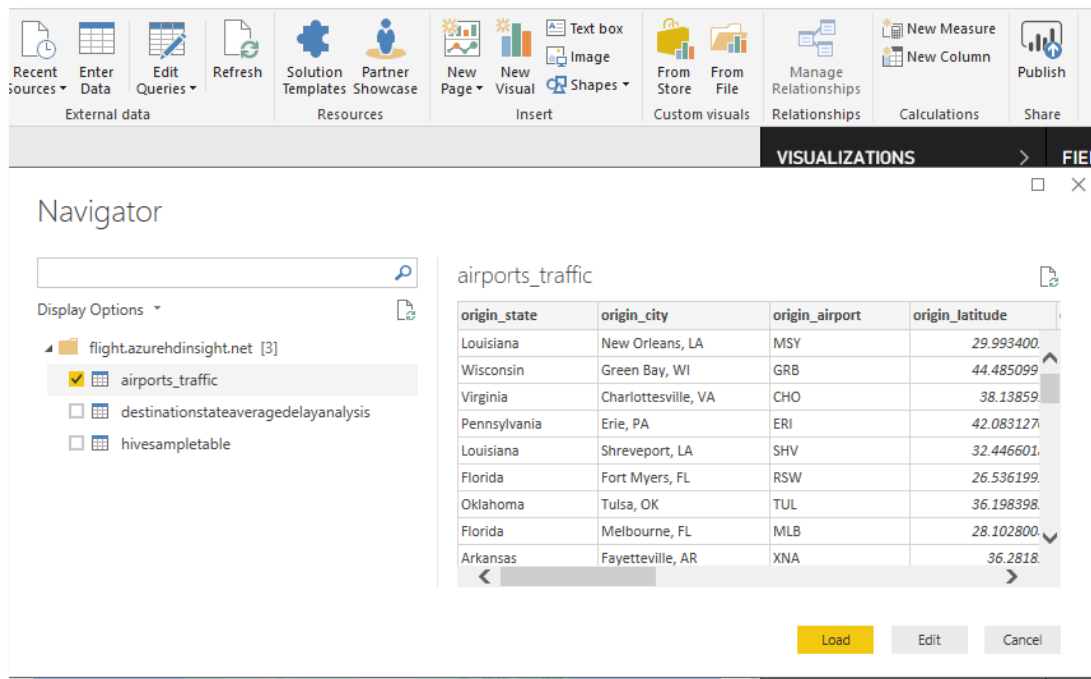
Output:

```
1 | spark.sql("show tables").show(false)
```

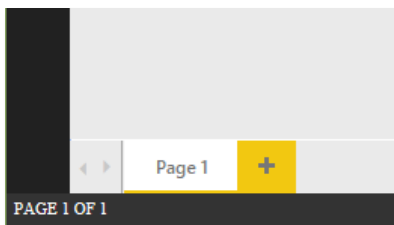
```
+-----+-----+-----+
|database|tableName|isTemporary|
+-----+-----+-----+
|default |airports_traffic|false|
|default |destinationstateaveragedelayanalysis|false|
|         |airports_na|true|
|         |departuredelays|true|
+-----+-----+-----+
```

8. Return on the Microsoft Power BI Desktop and click on the **Recent Sources** icon in the **Home** ribbon.

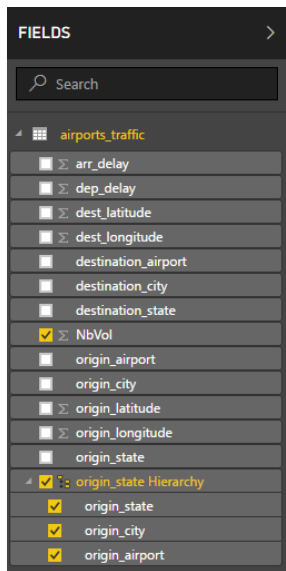
9. Select **spark clustername** sources, check the new **airports_traffic**, and push the **Load** button.



10. On your report you can observe the new table in the Fields panel named **airport_traffic**, add a **new page** in the bottom of the report and click on +



11. On your page 2, refactor your airports_traffic field panel: create a new hierarchy, drag and drop the **origin_city** on the **origin_state**, a new field named **origin_state Hierarchy** will be created, continue and add the **origin_airport** by drag and drop.



12. Add a matrix visualization, and add the origin_state_hierarchy as row and FlightCount as Value
 - a. Sort the matrix by **FlightCount** decreasing

Visualizations

Fields

\wedge airports_traffic

- ☐ Σ dep_delay
- ☐ Σ des_latitude
- ☐ Σ des_longitude
- ☐ destination_ai...
- ☐ destination_city
- ☐ destination_st...
- ☒ Σ FlightCount
- ☐ IATA
- ☐ origin_airport
- ☐ origin_city
- ☐ Σ origin_latitude
- ☐ Σ origin_longitu...
- ☐ origin_state

\wedge origin_state ...

- ☒ origin_state
- ☒ origin_city
- ☒ origin_airport

Rows

origin_state Hierarchy \vee \times

origin_state \times

origin_city \times

origin_airport \times

Columns

Add data fields here

Values

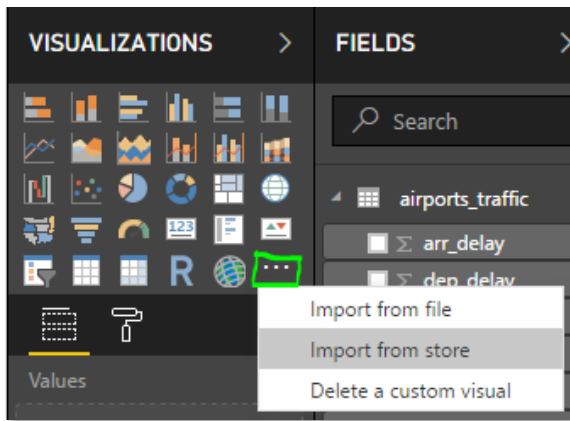
FlightCount \vee \times

Filters

origin_state	FlightCount
California	2866535
Los Angeles, CA	843904
San Francisco, CA	661159
San Diego, CA	308276
Oakland, CA	184585
San Jose, CA	170830
Sacramento, CA	165325
Santa Ana, CA	160711
Burbank, CA	87354
Ontario, CA	78871
Long Beach, CA	47229
Palm Springs, CA	39566
Fresno, CA	34055
Santa Barbara, CA	25732
San Luis Obispo, CA	14609
Monterey, CA	12855
Bakersfield, CA	10775
Arcata/Eureka, CA	7019
Redding, CA	3852
Total	22466964

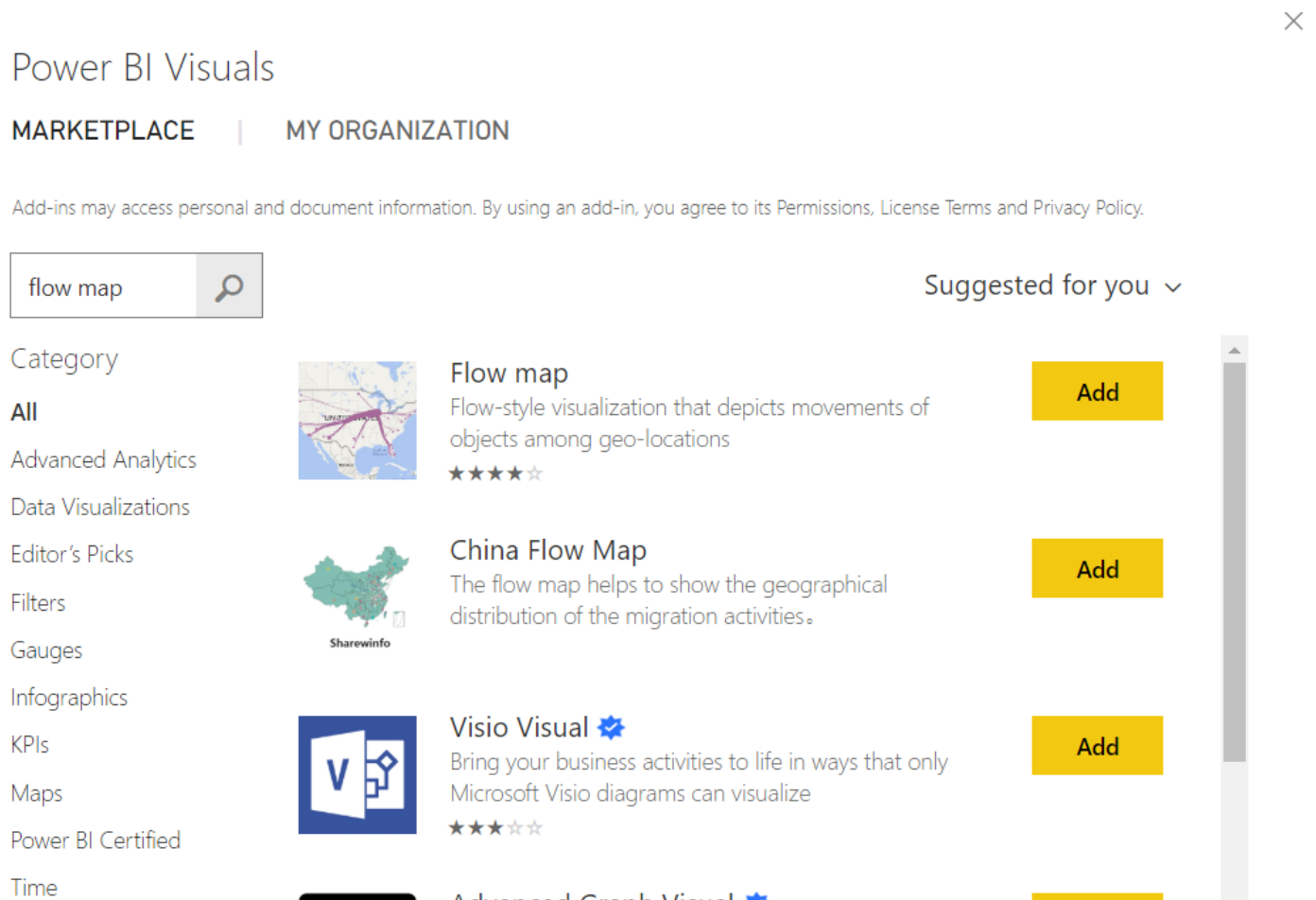
- b. Tips: you can expand the next level or only the next level on selected item, click on the **FlightCount** column to sort by the highest number of flight.

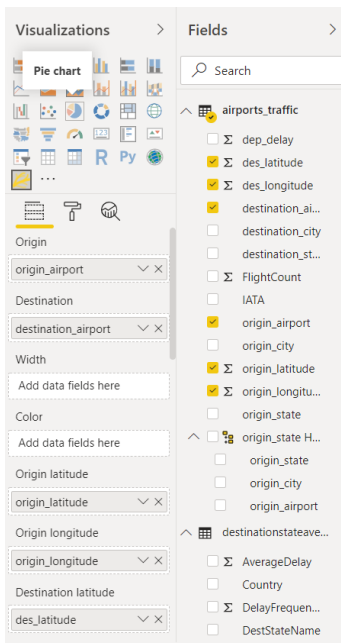
13. Add a new visualization from the store :



14. Select the ... and select Import from store.

15. When the Power BI Custom Visuals Store open, select the **Maps** category and choose the **Flow map** and Add.

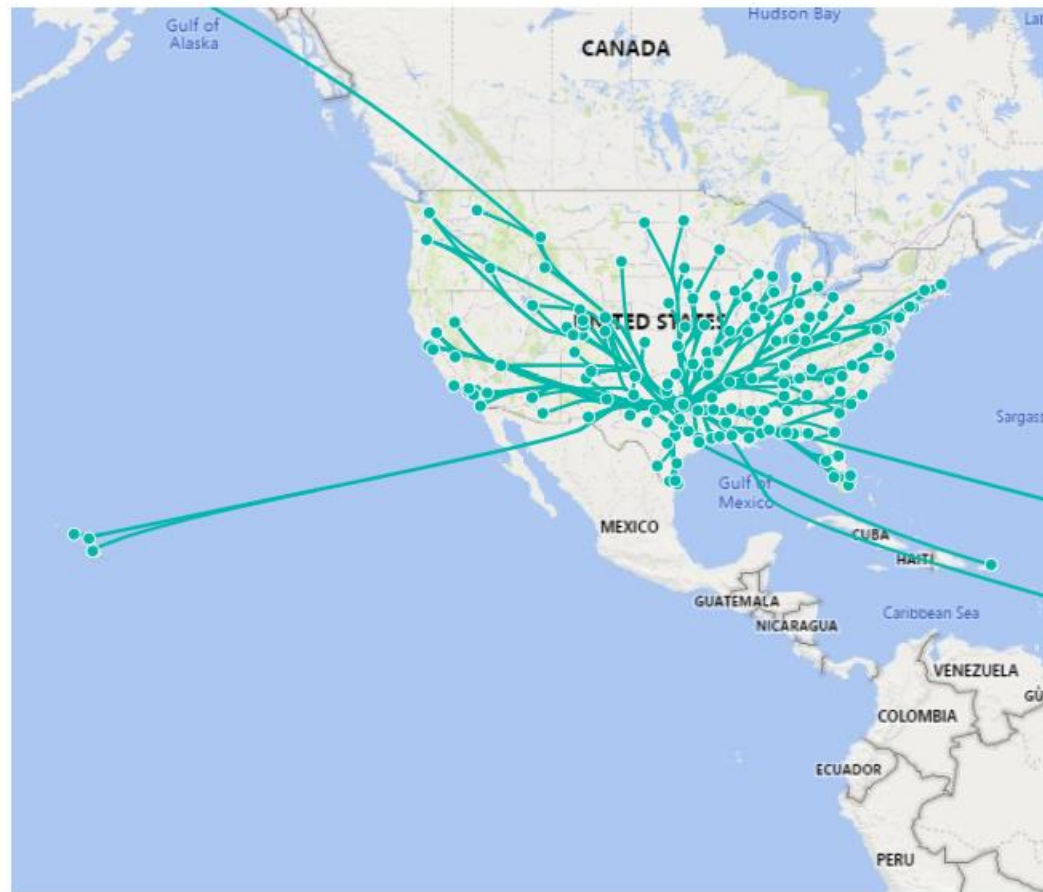




16. Add on the **map flow properties** and **place fields** as the snapshot
 - a. Drag & drop the field **origin_airport** to the Map flow's **Origin** property
 - b. Drag & drop the field **destination_airport** to the Map flow's **Destination** property
 - c. Drag & drop the field **flightCount** to the Map flow's **Value** property
 - d. Drag & drop the field average of **origin_latitude** to **Origin latitude**
 - e. Drag & drop the field average of **origin_longitude** to **Origin longitude**
 - f. Drag & drop the field average of **dest_latitude** to **Destination latitude**
 - g. Drag & drop the field average of **dest_longitude** to **Destination longitude**

17. Select an **origin_city** in the **matrice**. You should have something similar to this:

origin_state	FlightCount
California	2866535
Texas	2495260
Florida	1741372
Georgia	1527215
Illinois	1484963
New York	1009830
Colorado	954763
Arizona	699453
North Carolina	641966
Nevada	634033
Virginia	594539
Michigan	580694
Washington	544861
Minnesota	510395
Massachusetts	471634
New Jersey	465356
Utah	438512
Pennsylvania	429721
Missouri	408542
Hawaii	392036
Maryland	373662
Tennessee	323110
Ohio	297825
Louisiana	269717
Oregon	262624
Wisconsin	210167
Indiana	159413
Kentucky	145258
Alaska	141217
Oklahoma	132806
South Carolina	121319
Puerto Rico	111393
Alabama	105262
Total	22466964



Disclaimer: Once you have completed the lab, to reduce costs associated with your Azure subscription, you may want to delete your clusters!!!!

Terms of use

© 2019 agileDSS. All rights reserved.

By using this hands-on lab, you agree to the following terms:

The technology/functionality described in this hands-on lab is provided by agileDSS for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the hands-on lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this hands-on lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCTIONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.