Microsoft Machine Learning in a Day Workshop

Student Guide

September 2018

Contents

Machine Learning in a Day student guide

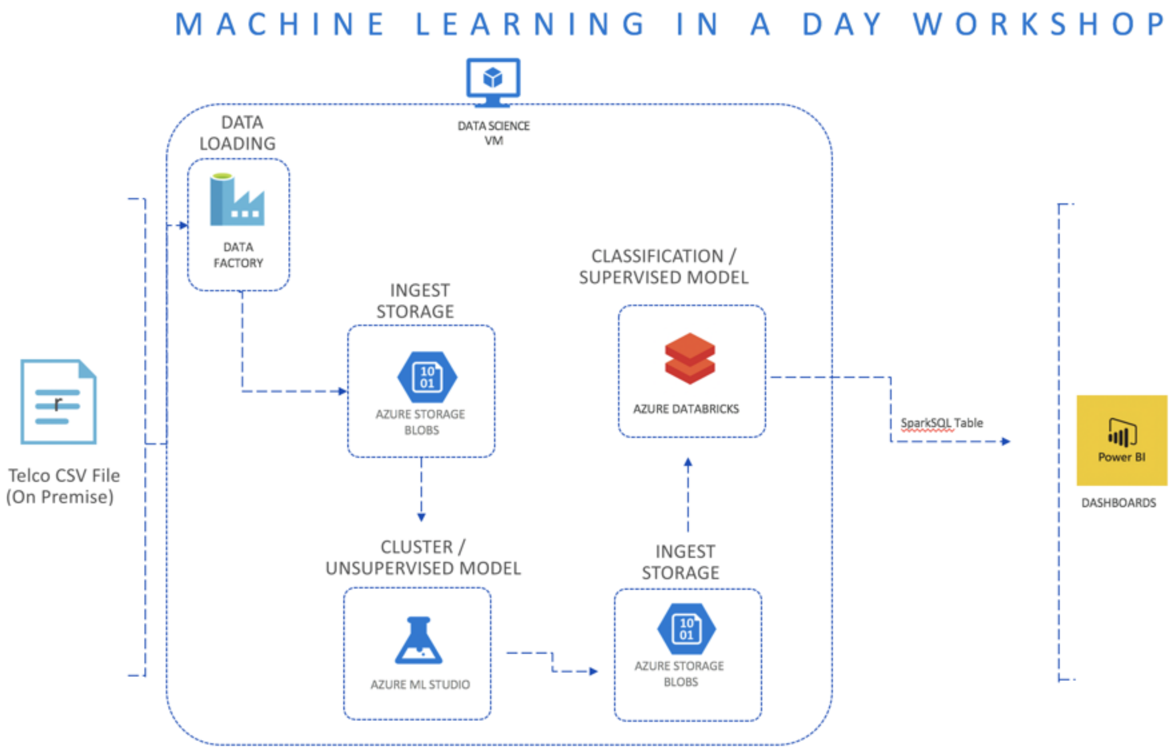
Abstract and learning objectives

In this workshop, you will complete will build a Machine Learning model that will be used to predict whether customers remain with business or move onto other services based on their behavior, demographics, and spending. You will leverage several Microsoft products and services on Microsoft Azure to deliver the Machine Learning Model.

The dataset that will be used in the workshop contains the following information

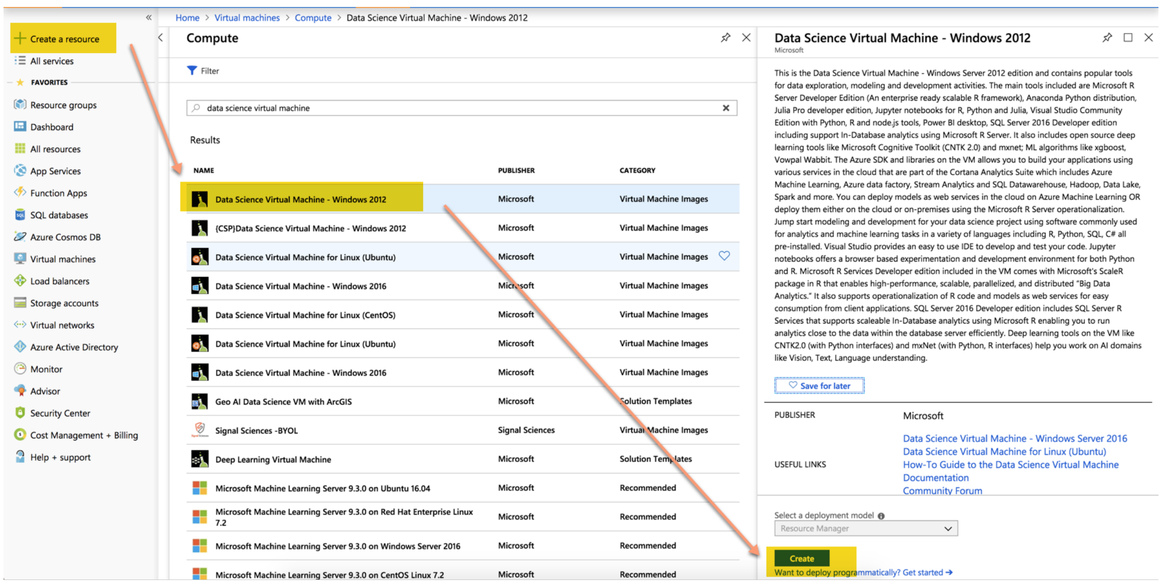
1. Customers who left within the last month – the column is called **Churn**
2. Services that each customer has signed up for
    a. Phone
    b. Multiple lines
    c. Internet
    d. Online security
    e. Online backup
    f. Device protection
    g. Tech support
    h. Streaming TV and movies
3. Customer account information
    a. How long they've been a customer
    b. Contract
    c. Payment method
    d. Paperless billing
    e. Monthly charges
    f. Total charges
4. Demographic information about customers
    a. Gender
    b. Age range
    c. If they have partners and dependents

Architecture

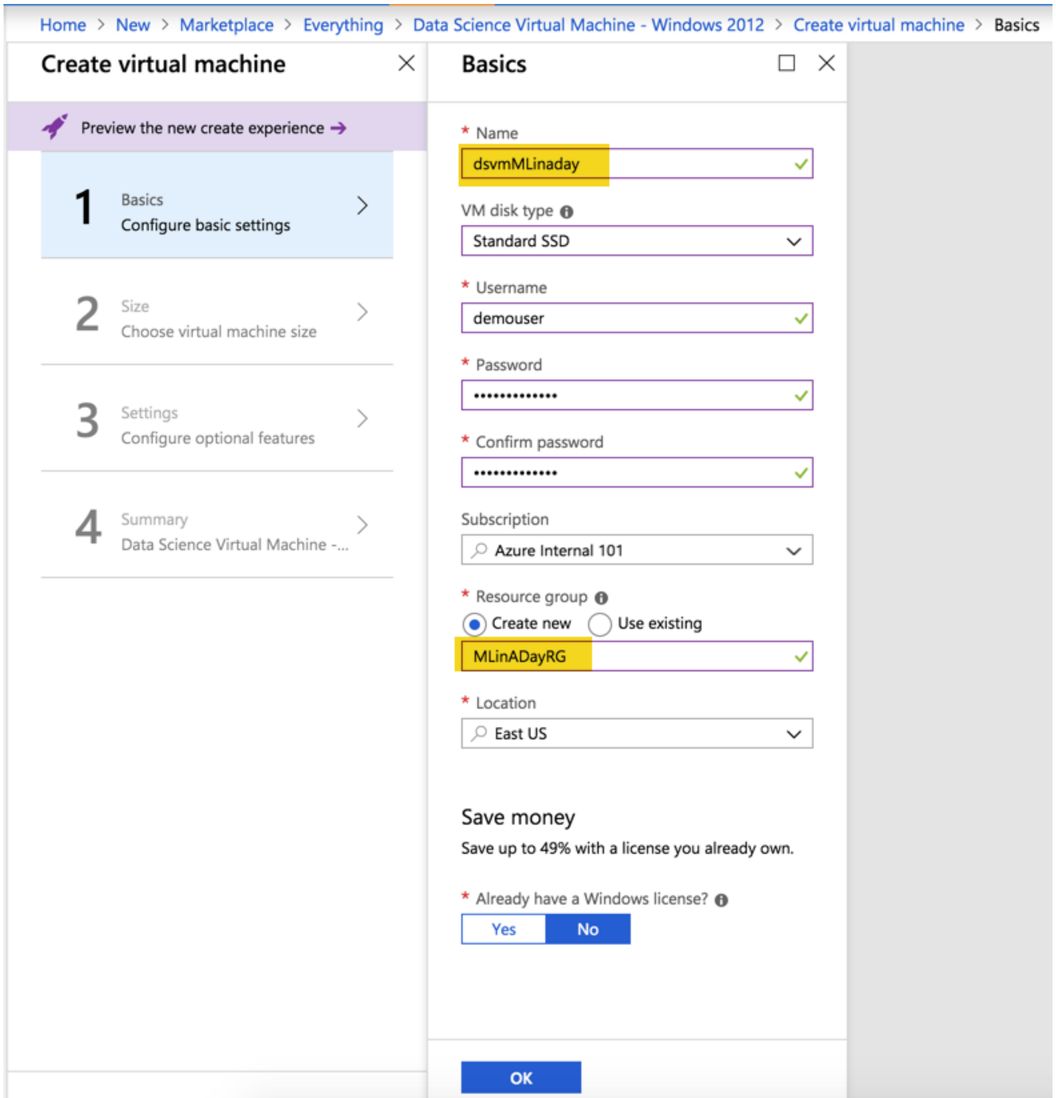


Section 1: Set up a Data Science Virtual Machine (DSVM) on Azure Portal

1. The use of this documentation requires the subscription of a Microsoft Azure account either from a personal email account or a work email account.
2. Visit www.portal.azure.com and click on **Create a resource** on the left-hand side and type in **Data Science Virtual Machine – Windows 2012** and select the **Create** button as seen in the following screenshot:

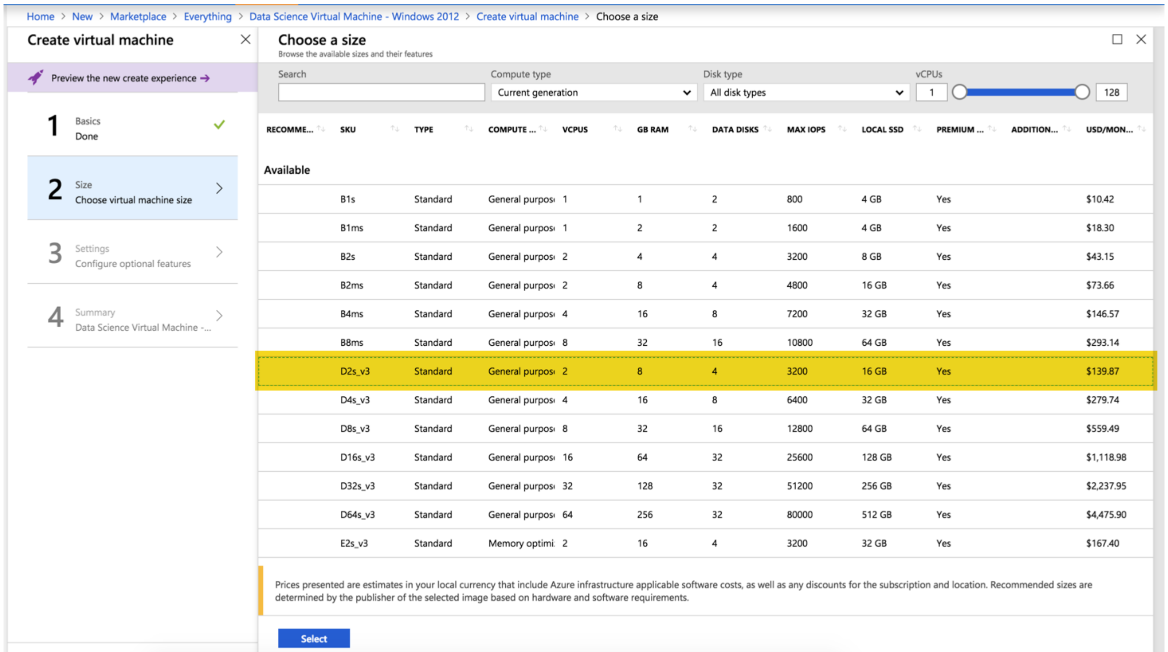


3. Assign the following basic configuration settings for your Data Science Virtual Machine and the select the **OK** button as seen in the following screenshot:

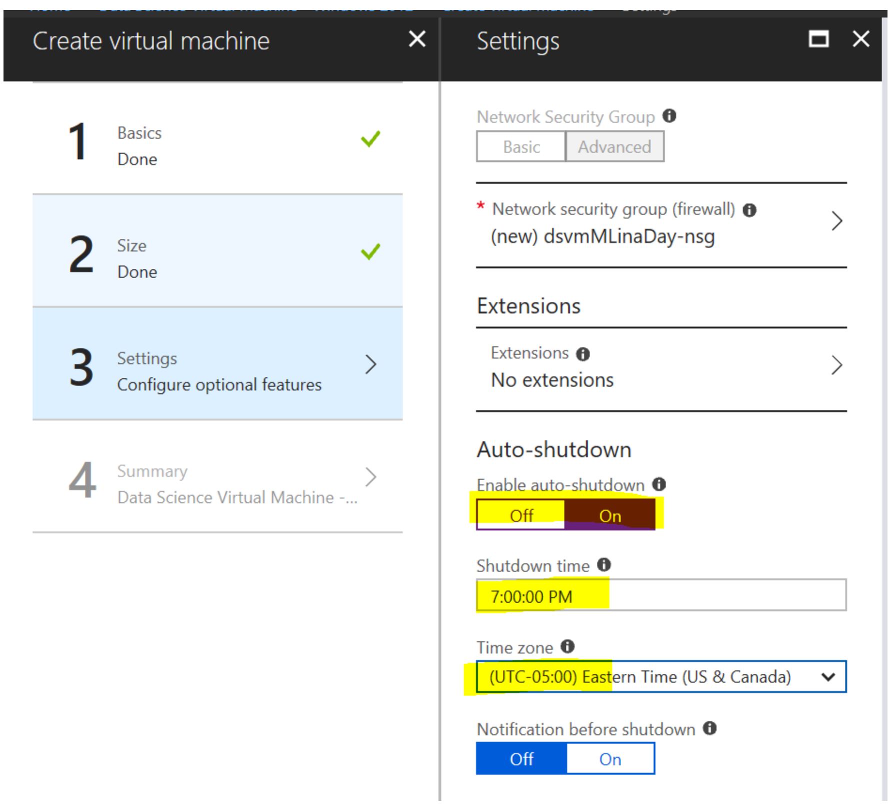


Please note that you can keep your Names and passwords different from this document as long as you remember what you used for future use. In this section, we will assign a username of **demouser** and a password of **#MLinaDay20!8** (Please keep track of both the Username and Password as you will need both of them several times as we go through future sections of this workshop).

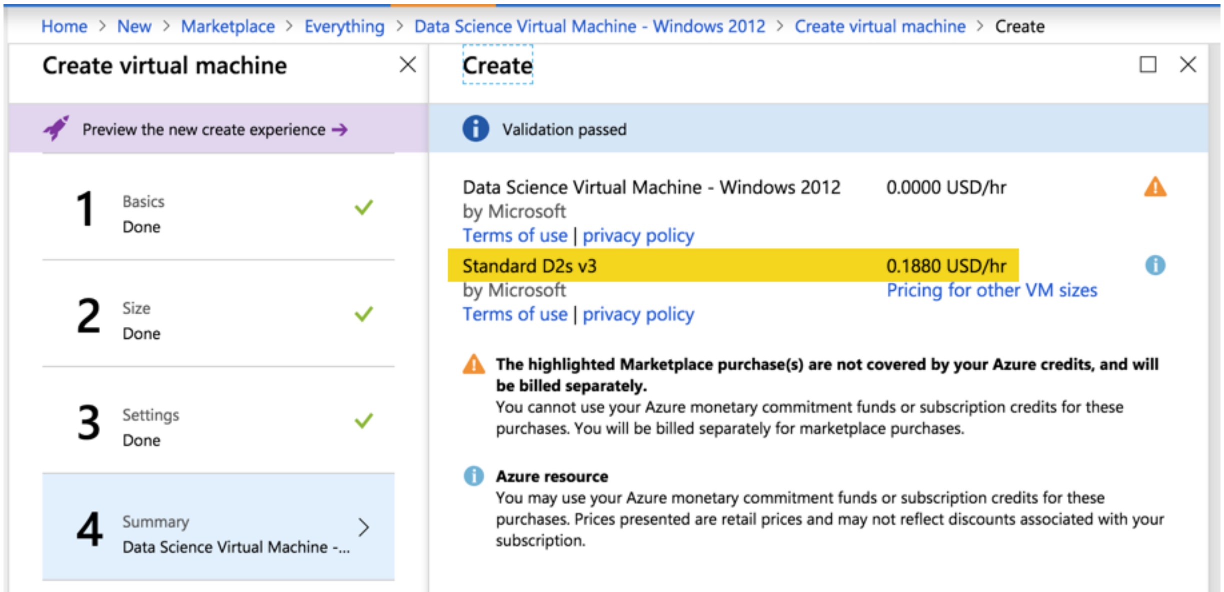
4. Choose the following size VM (**D2S_v3**) for the purposes of this workshop as seen in the following screenshot:



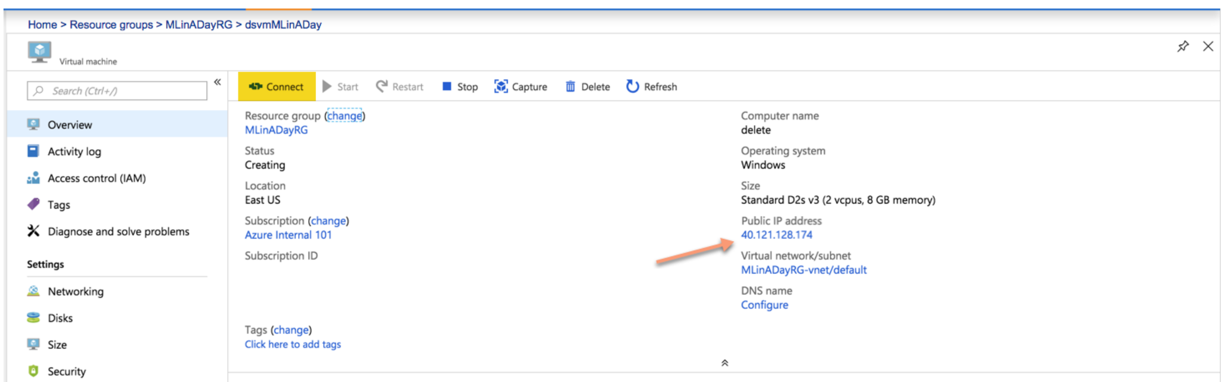
5. For settings, the only configuration that is truly needed is to enable on **auto-shutdown** every evening at **7pm EST** (your manager will thank you later on !!!!) and then select **OK** as seen in the following screenshot:



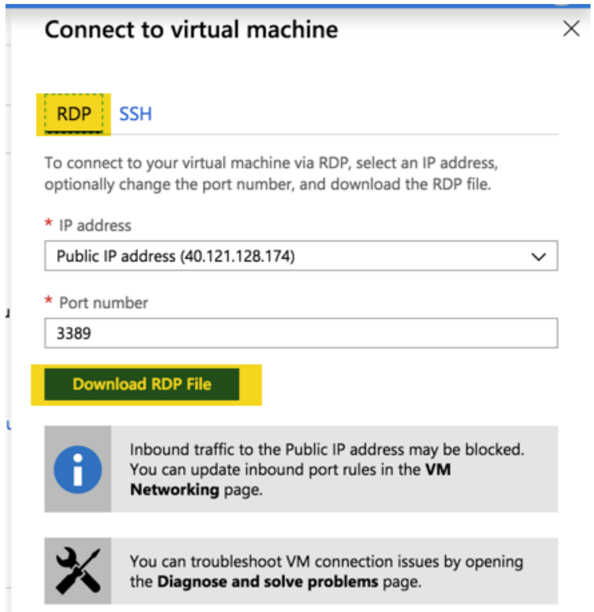
6. Click **OK** on the summary section and confirm everything that was selected is accurate and let the provisioning process of the DSVM begin by clicking on **Create** as seen in the following screenshot:



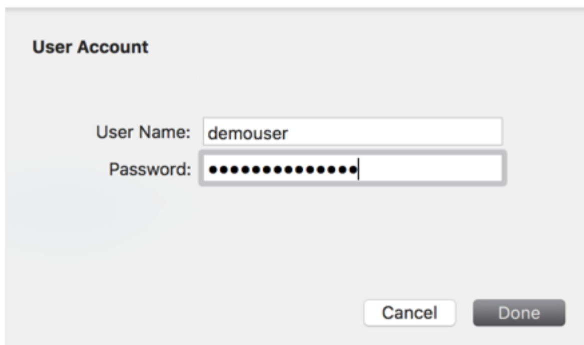
7. Once the provisioning is complete, you can go to your resource and view your DSVM overview and obtain your **Public IP address** as seen in the following screenshot:



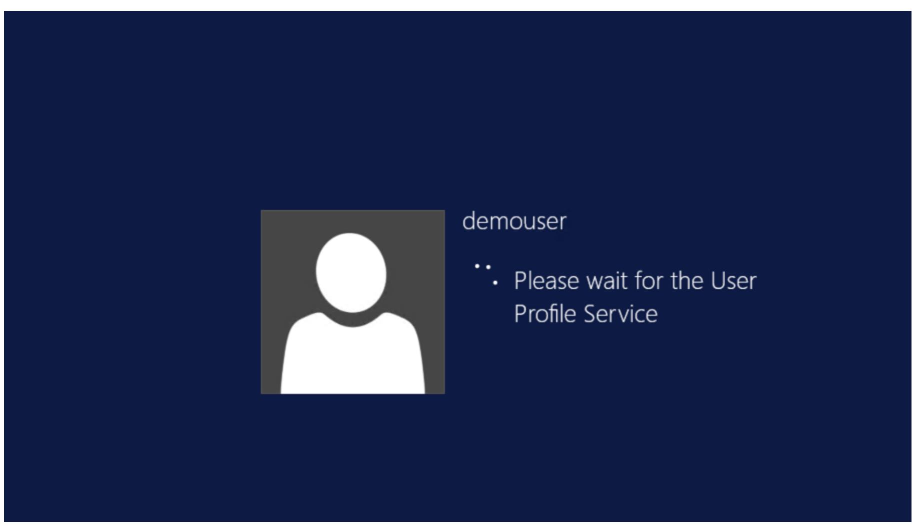
8. Make sure that your DSVM is currently running and click on the **Connect** button to enter your virtual machine using **RDP** as seen in the following screenshot:



9. Enter your login credentials that were created back in Step # 3 as seen in the following screenshot:



10. It may take a few extra moments while your profile is being created and you will see the following window:

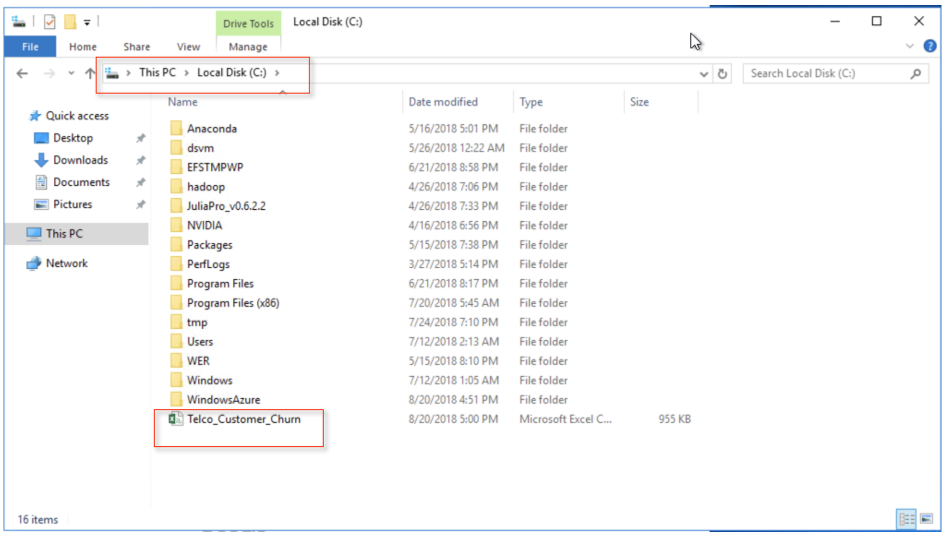


11. Once you log in using your credentials using the RDP approach, you should see the following remote server for the data science virtual machine as seen in the following screenshot:

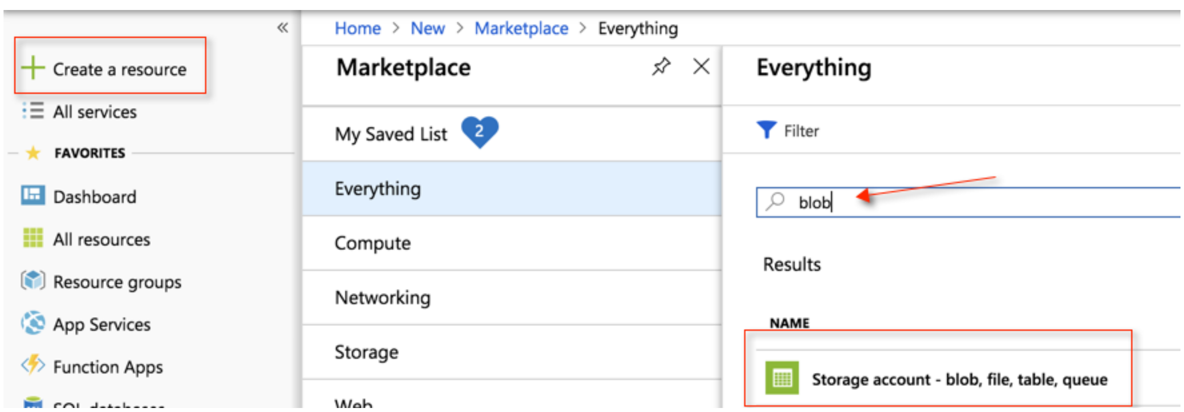

Recycle Bin
Juno for JuliaPr...
Visual Studio 2017
Analytics Labs
Jupyter Notebo...
Visual Studio Code
Azure Machi...
Microsoft Azure Po...
Weka 3.8
Azure SDK Docs
Microsoft SQL Server ...
Discover Your DSVM
Microsoft Web Pl...
DSVM Version
Microsoft-...
Evince PDF Reader
Power BI Desktop
Firefox
RStudio
Get Started Datascien...
Sample Gallery
JuliaPro - Comma...
Spyder
Deployment Id: 879154d8364d46729fc9722bb15ff6f4
Internal IP: 100.113.34.89
Public IP: 13.68.113.77
Boot Time: 9/5/2018 3:20 PM
Free Space: C:\ 60.96 GB NTFS
Memory: 8192 MB
OS Version: Windows Server 2012 R2 Datacenter
User Name: demouser
Windows Server 2012 R2

Section 2: Download Telco dataset and create Blob Storage Account and Container access

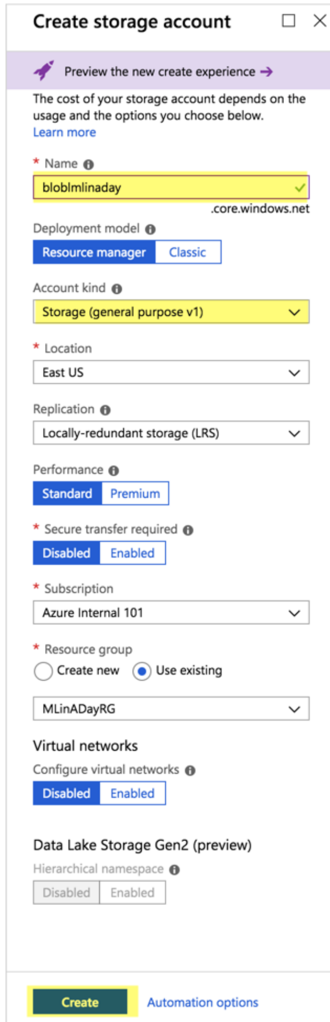
1. The Telco Churn Dataset (**TELCO_CUSTOMER_CHURN.CSV**) can be downloaded from the following link: https://blobmlinadayahmed.blob.core.windows.net/datasources/Telco_Customer_Churn.csv
2. For now, the dataset can be saved in the C:\ drive as seen in the following screenshot:



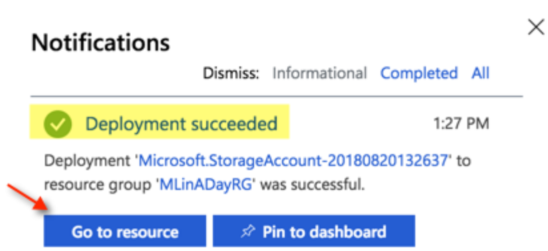
3. Next we will return to our Azure portal (https://portal.azure.com) and create a Blob storage account to house our data. We can do this by selecting **Create A Resource**, typing **Blob** in the search, and then selecting **Storage Account – blob, file, table, queue** as seen in the following screenshot:



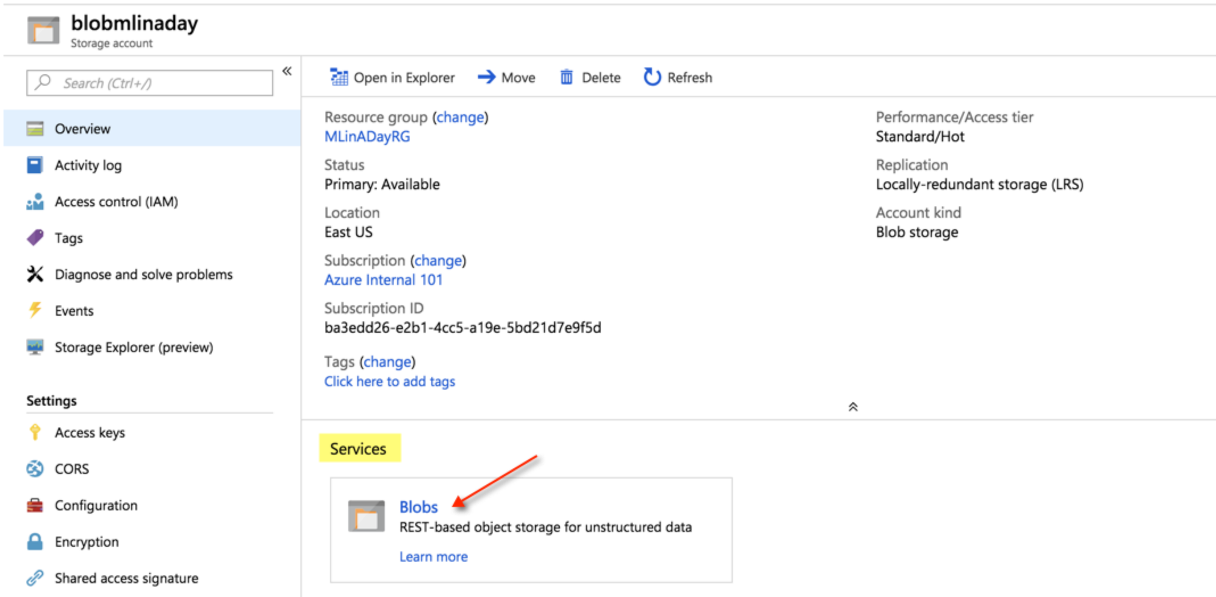
4. Once you create a **Blob** account, you can name and customize the properties of the account as the following, for optimal performance in this workshop (Please note than any existing resource that is created from this point forward in the workshop should ideally use the same existing **Resource Group** that we created in Section 1):



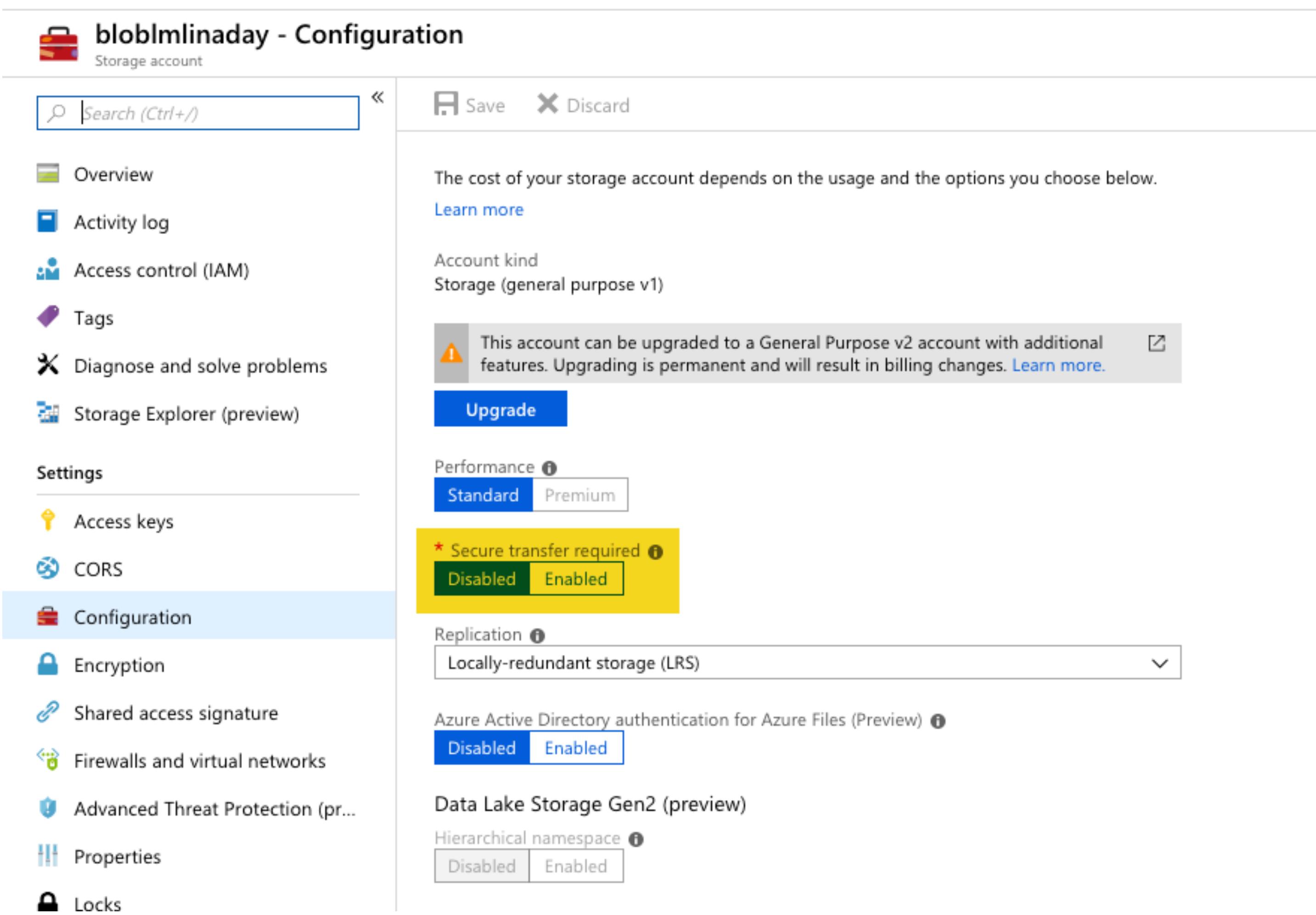
5. Once the Blob account has been successfully deployed, we can go to the storage account by clicking on **Go to Resource**, as seen in the following screenshot:



6. Once inside the resource, select **Blobs** under the **Services** section as seen in the following screenshot:

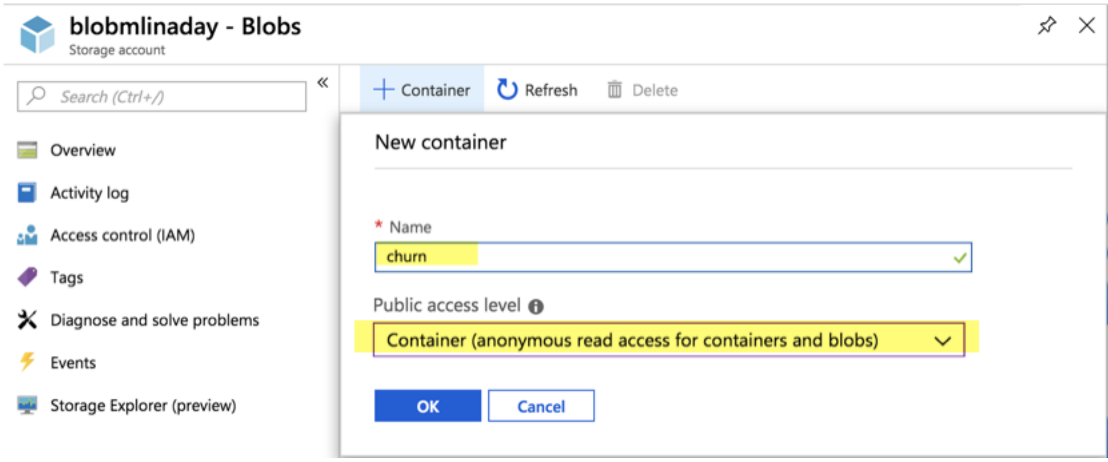


7. ****Important Step ***** Ensure that your blob account has the following setting configured to ensure that **Secure Transfer Required** is **Disabled**.

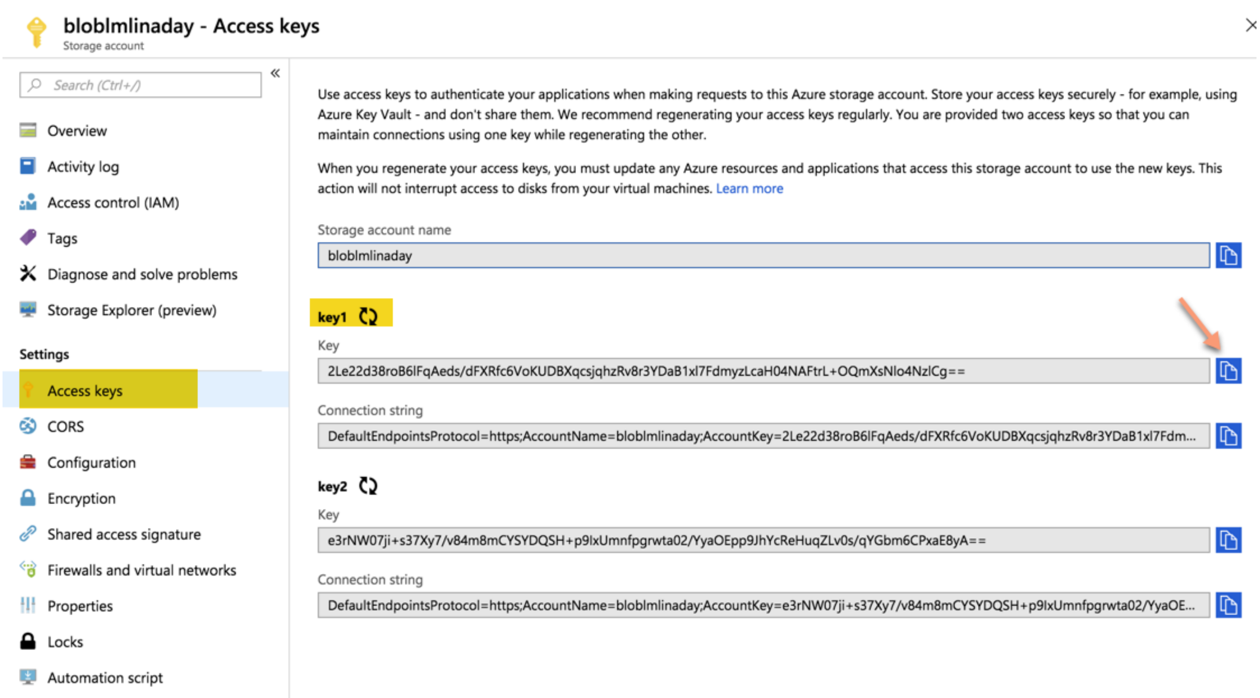


8. We should now be in the Containers section; however, if this is the first time we are entering the Container, we should see a message that says **You don't have any containers yet. Click '+ Container' to get started**.

9. Go ahead and create a new Container called **Churn** and set **Public access level** to **Container (anonymous read access for containers and blobs)**, as seen in the following screenshot:



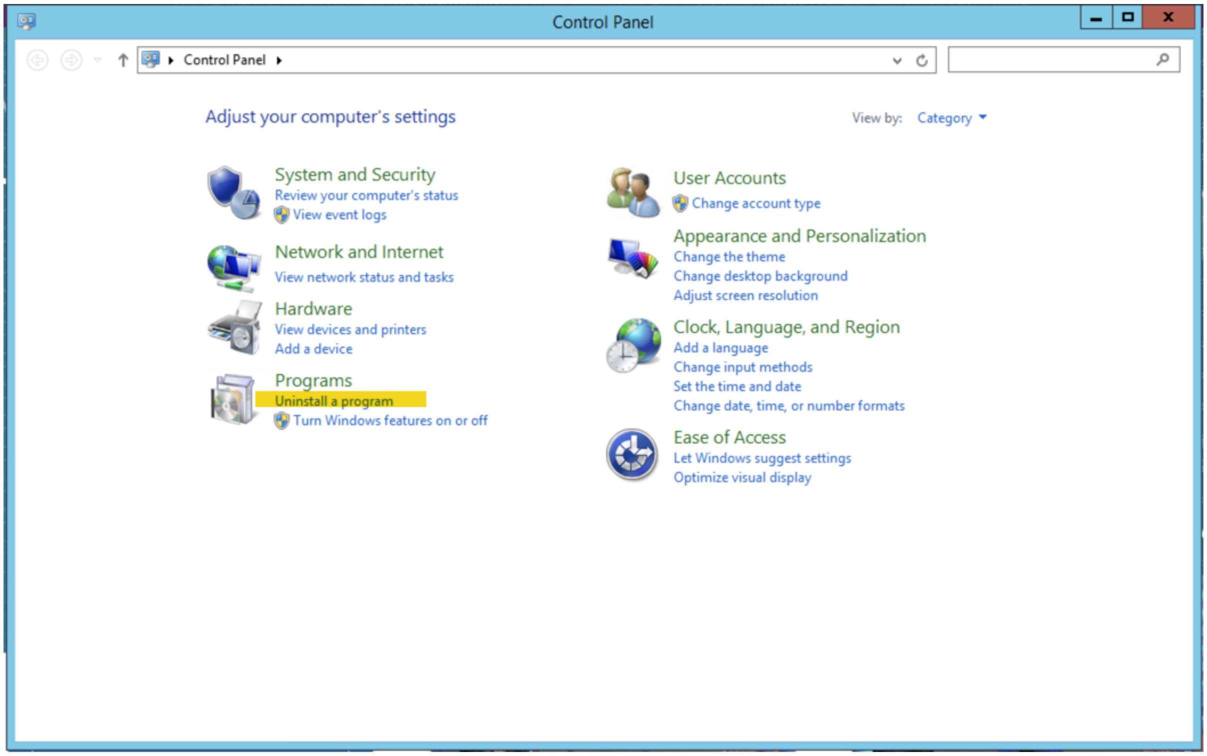
10. The only thing that we will need right now is an **Access Keys** that will be used in future sections of this workshop. We can copy **key1** and paste it somewhere handy in a notepad editor as seen in the following screenshot:



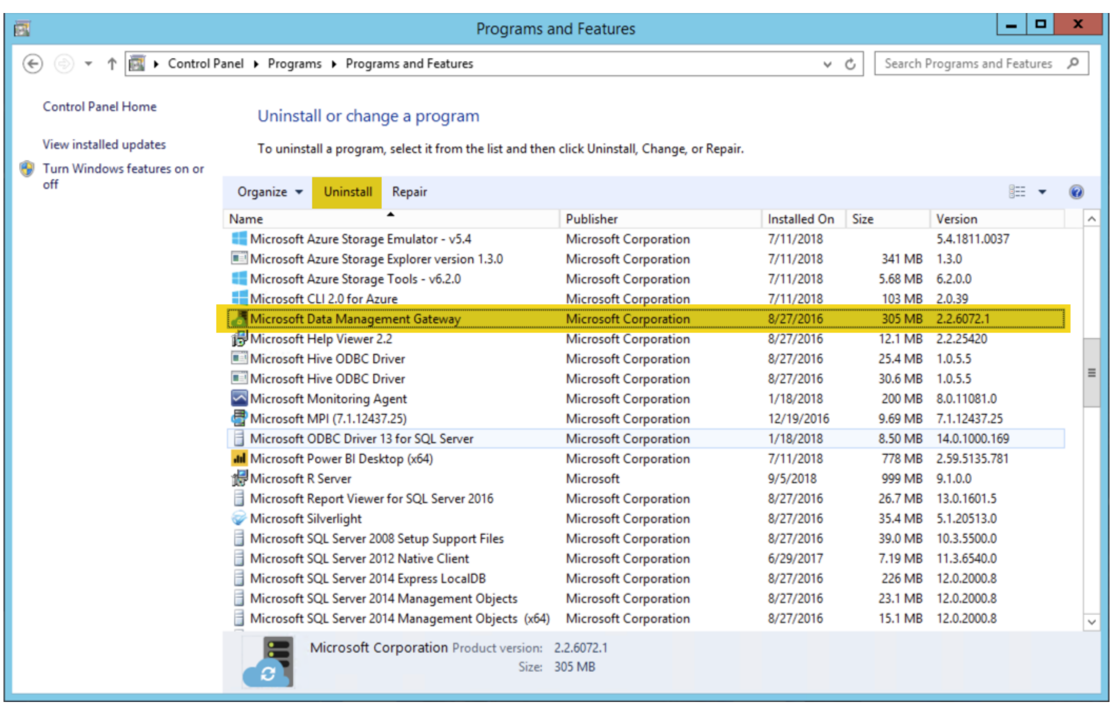
11. Once the container has been successfully created, we can move onto the next section of adding data to the container using an orchestration/ETL tool called Azure Data Factory.

Section 3: Upload CSV file to Blob Storage Container with Azure Data Factory V2.

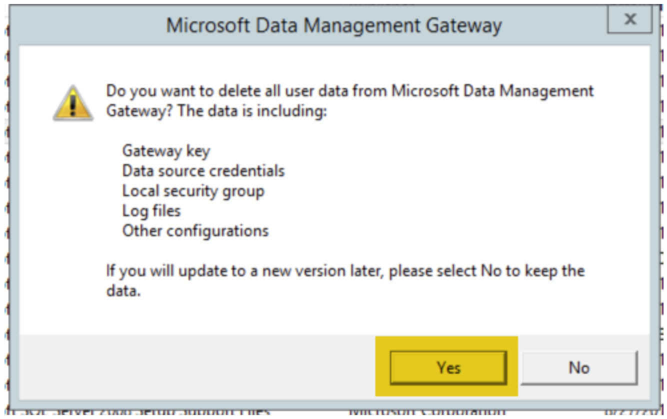
1. You have now completed the provisioning of a data science virtual machine on Windows Server 2012. One note to factor for this virtual machine is we will uninstall a product that is outdated and download later on in this section. First, we will go to the **Control Panel** in the DSVM and select to **Uninstall A Program** as seen in the following screenshot:



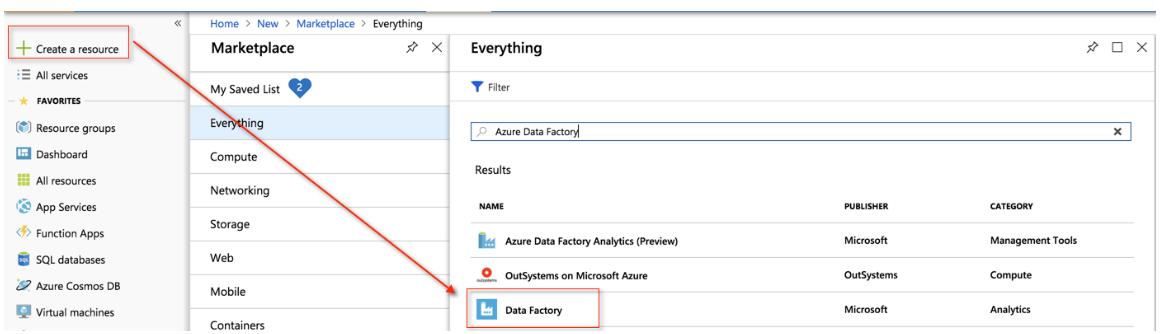
2. Next we will go ahead and **Uninstall Microsoft Data Management Gateway** as seen in the following screenshot:



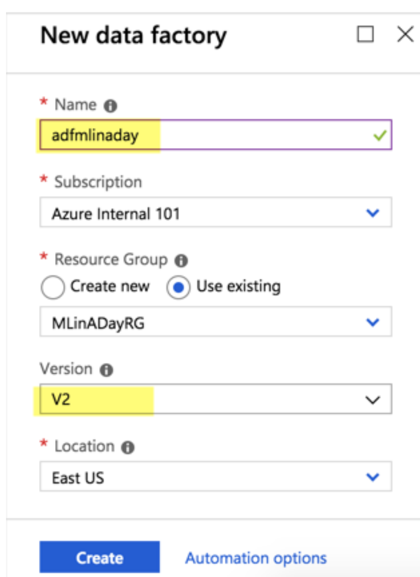
3. Select **Yes** to get rid of all data as seen in the following screenshot:



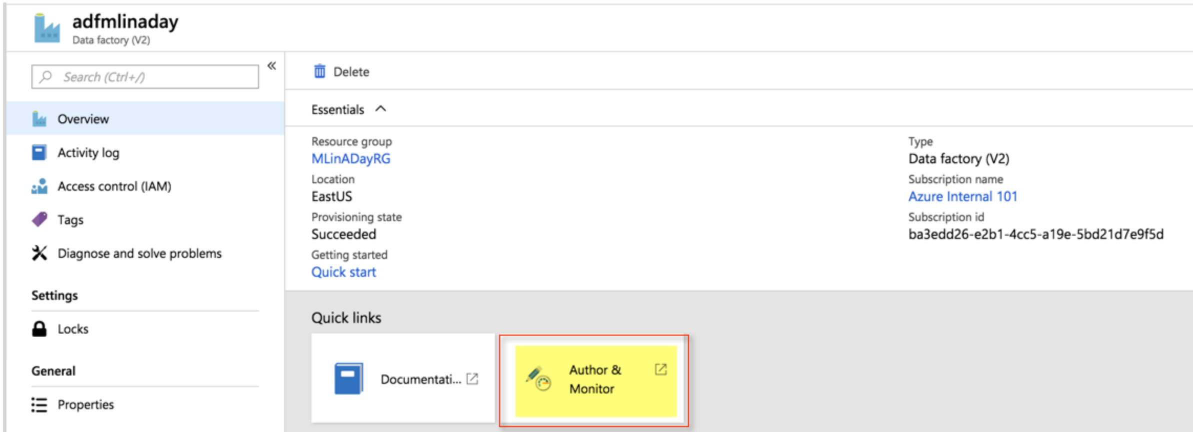
4. As we move forward in this section, we will require accessing the Azure Portal (https://portal.azure.com) inside of a browser within the data science virtual machine to set up **Azure Data Factory V2.**
5. Once again, as we did with the Blob Storage as well as with the Data Science Virtual Machine, we are ready to create a new resource. This time we will build an orchestrator that will create a job to move our CSV file from on-premise on our Virtual Machine to our storage in Blob on Azure. The next step is to create a resource for **Azure Data Factory** as seen in the following screenshot:



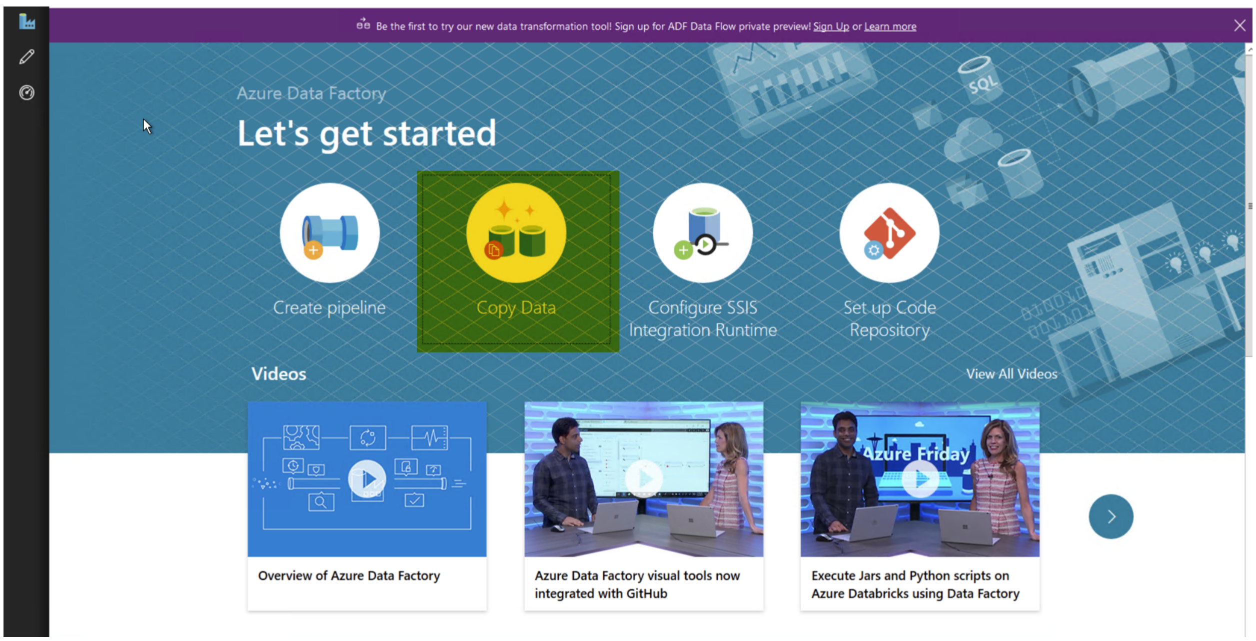
6. Once the resource is created, we will want to specify the following options and then click on **Create**:



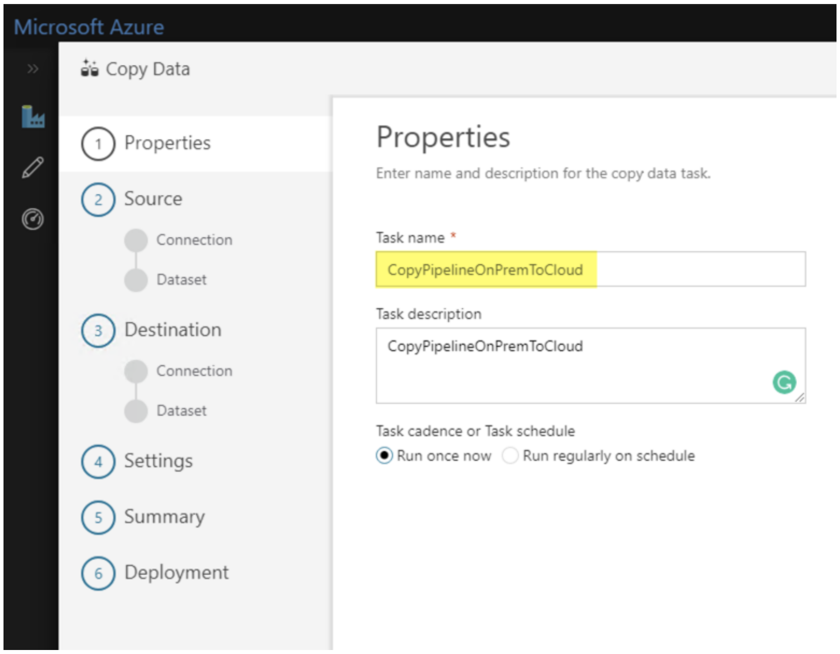
7. Once the resource is completed, we can go to the resource and click on the icon **Author & Monitor** as seen in the following screenshot:



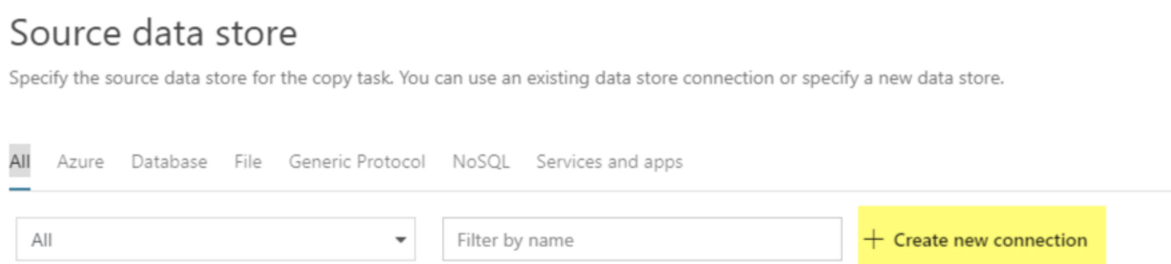
8. We are now in the landing page for Azure Data Factory. The next step is to click on the second icon, **Copy Data**:



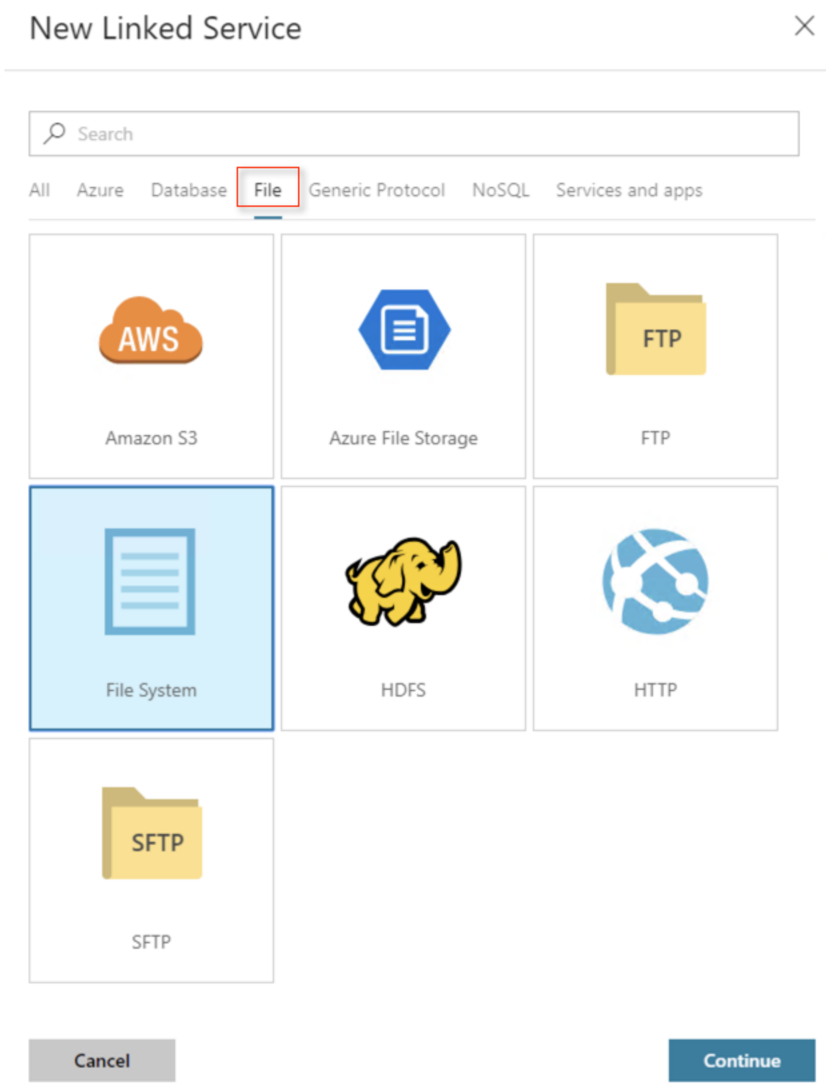
9. Our first task is to enter name and description for the copy data task as seen in the following screenshot:



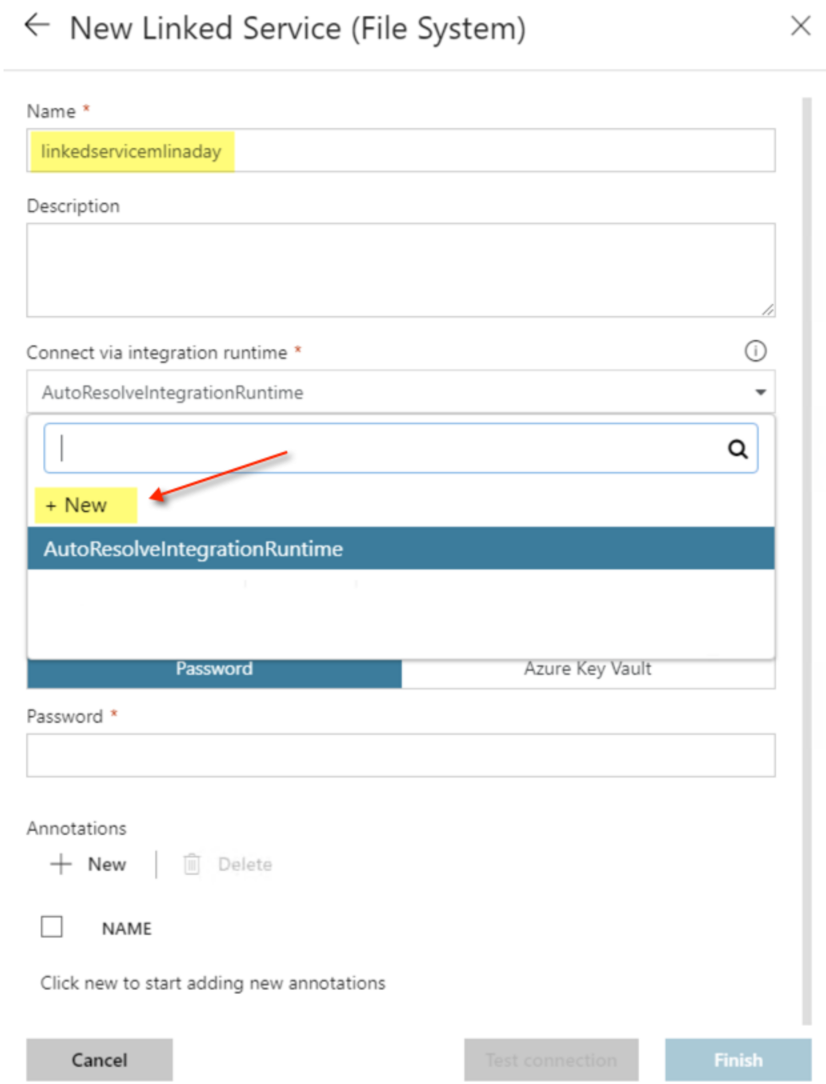
10. We named the task **CopyPipelineOnPremToCloud**. The next step is to specify a data source. Since this source will be on-premise on our virtual machine, we will need to set up **+ Create a new connection** as seen in the following screenshot:



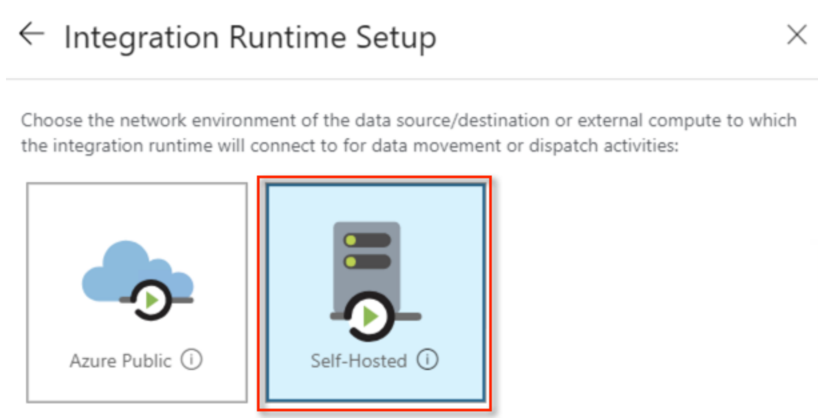
11. Select **File** and then **File System** for a **new Linked Service** between on-premise and in Azure. Then select **Continue** as seen in the following screenshot:



12. Specify a name for the Linked Service such as **linkedservicemlinaday** as well as clicking on a **+New integration runtime** as seen in the following screenshot:

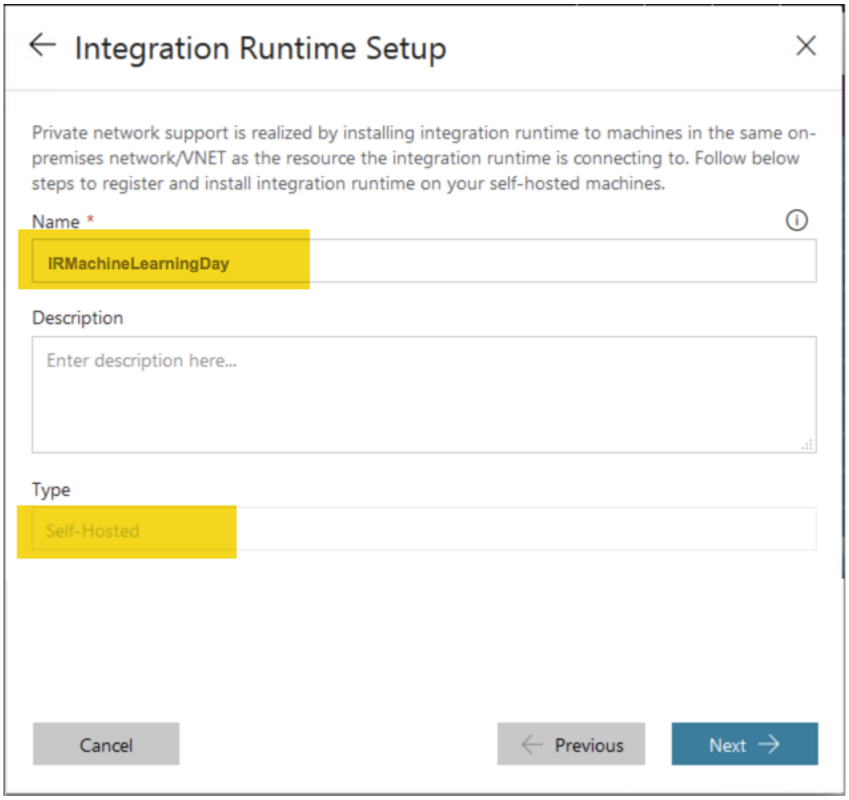


13. Select a **Self-Hosted** setup for the integration runtime as seen in the following screenshot:

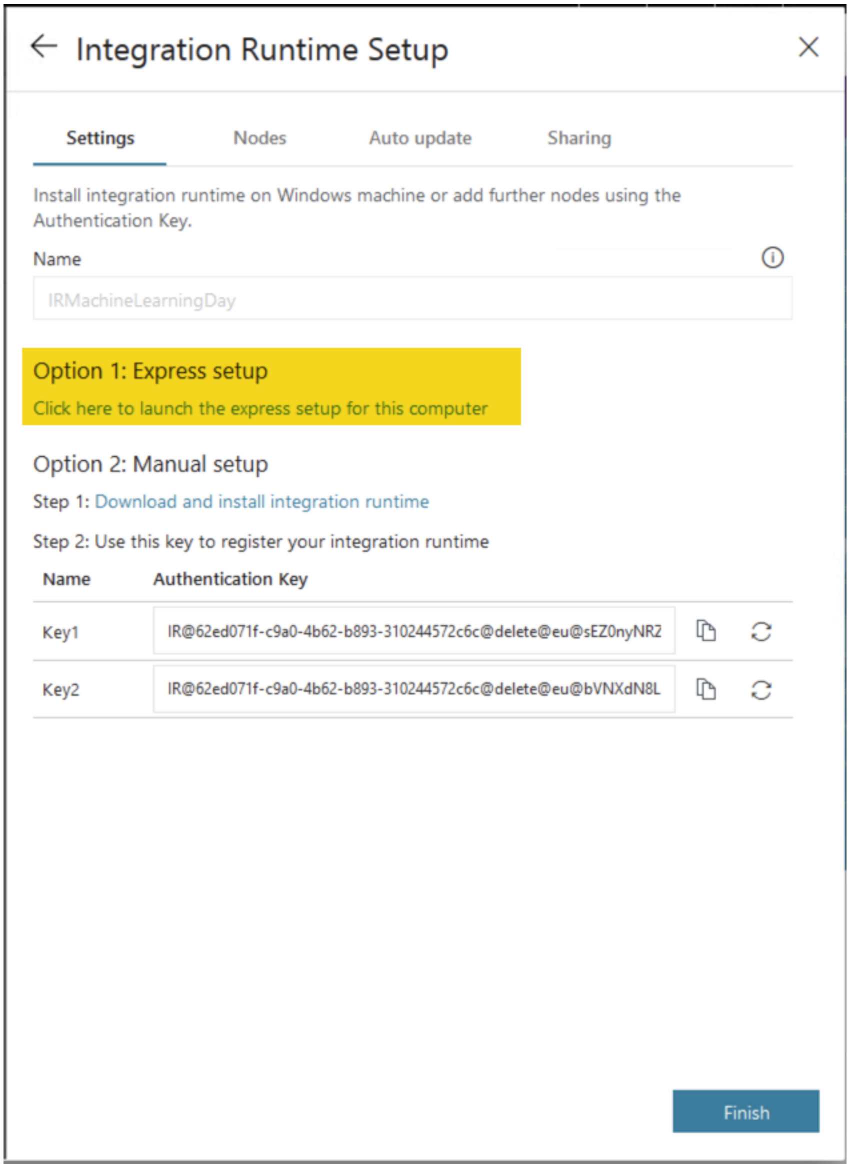


Choose an **Azure Public** environment if you are accessing services with public accessible endpoints. Choose **Self-Hosted** if you are accessing services in a private network like your on-premises environment, or in Virtual Network Environments like Azure Virtual Network. For our purposes, we will go with Self-Hosted.

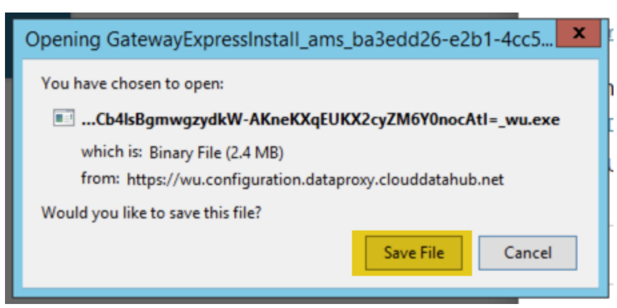
14. Private network support is realized by installing integration runtime to machines in the same on-premises network/VNET as the resource the integration runtime is connecting to. Follow below steps to register and install integration runtime on your self-hosted machines with a name called **IRMachineLearningDay** and then select **Next** as seen in the following screenshot:



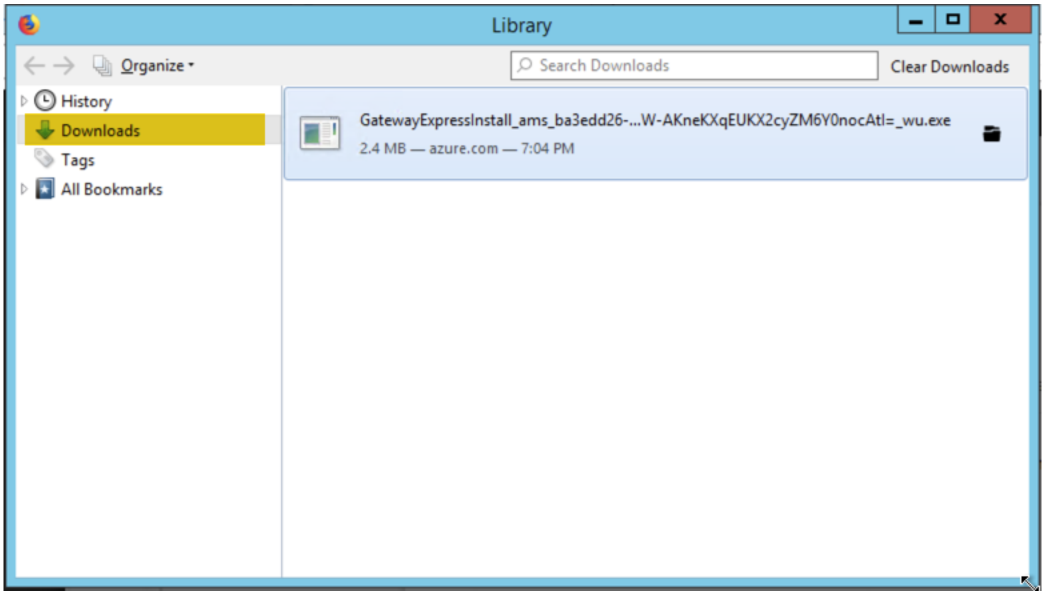
15. Select **Option 1: Express Setup** for the Integration Runtime as seen in the following screenshot:



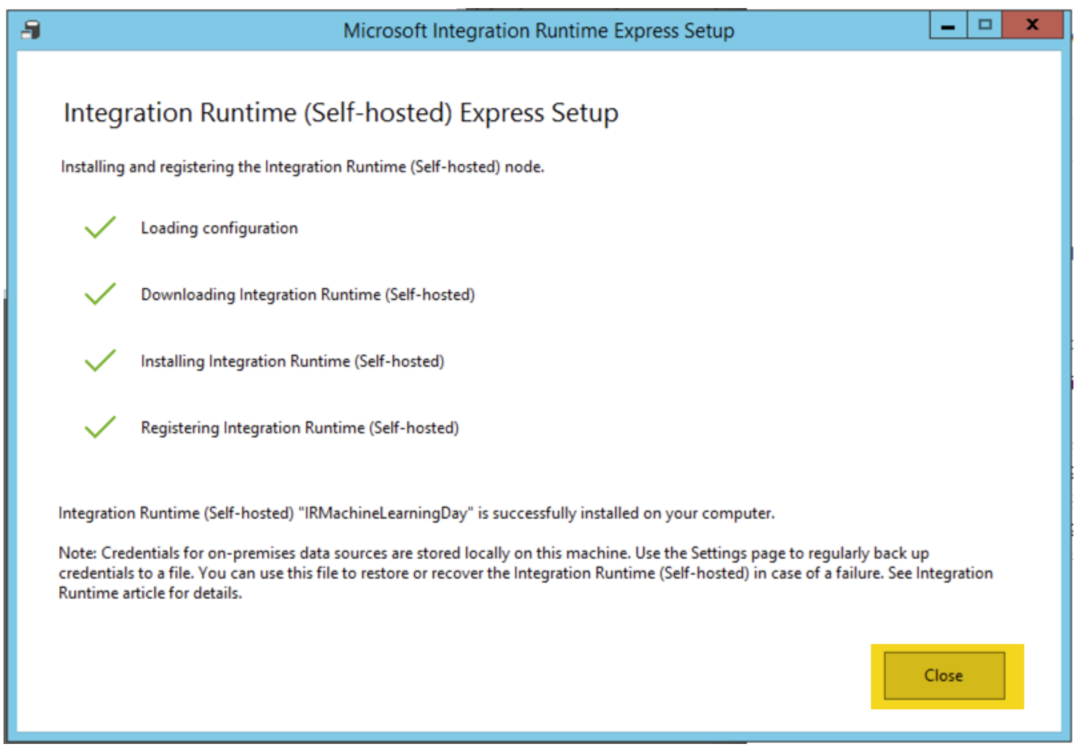
16. Save the Express Setup File (usually it will default to the Downloads folder) as seen in the following screenshot:



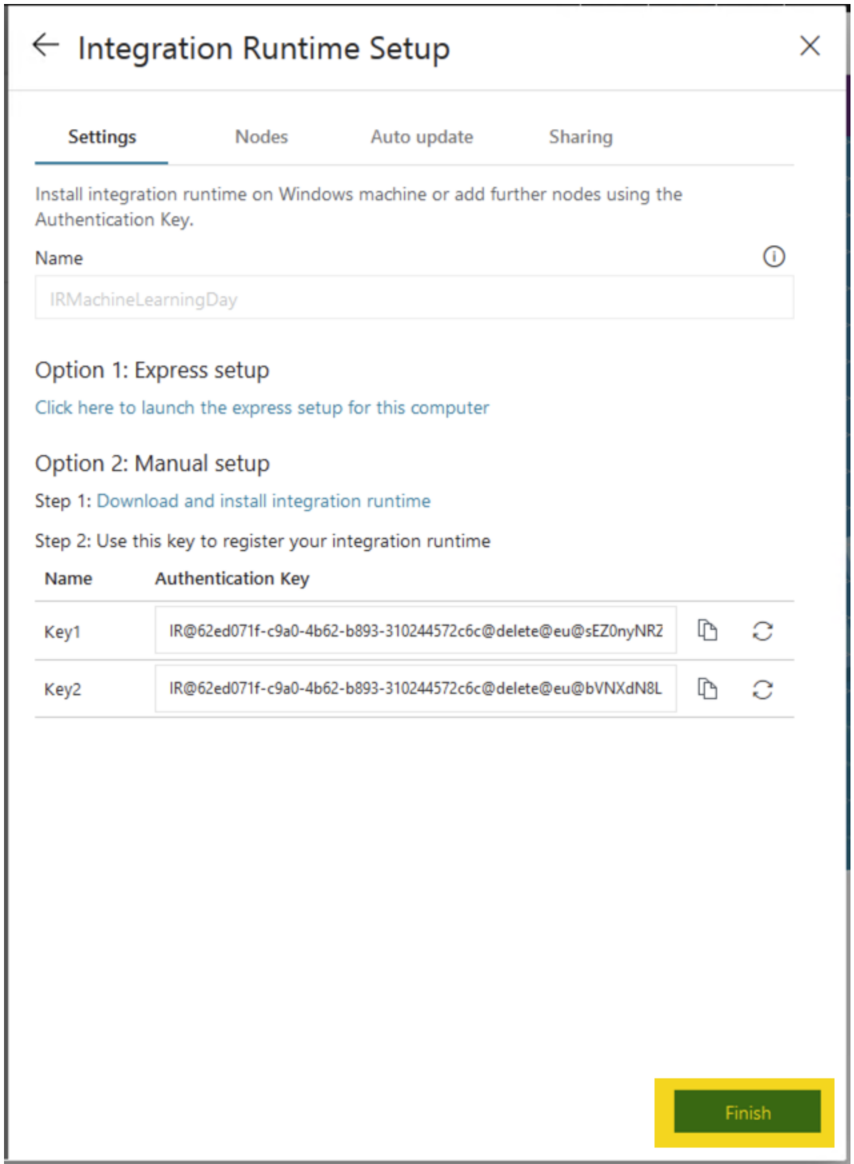
17. Execute the File from the **Downloads** folder as seen in the following screenshot:



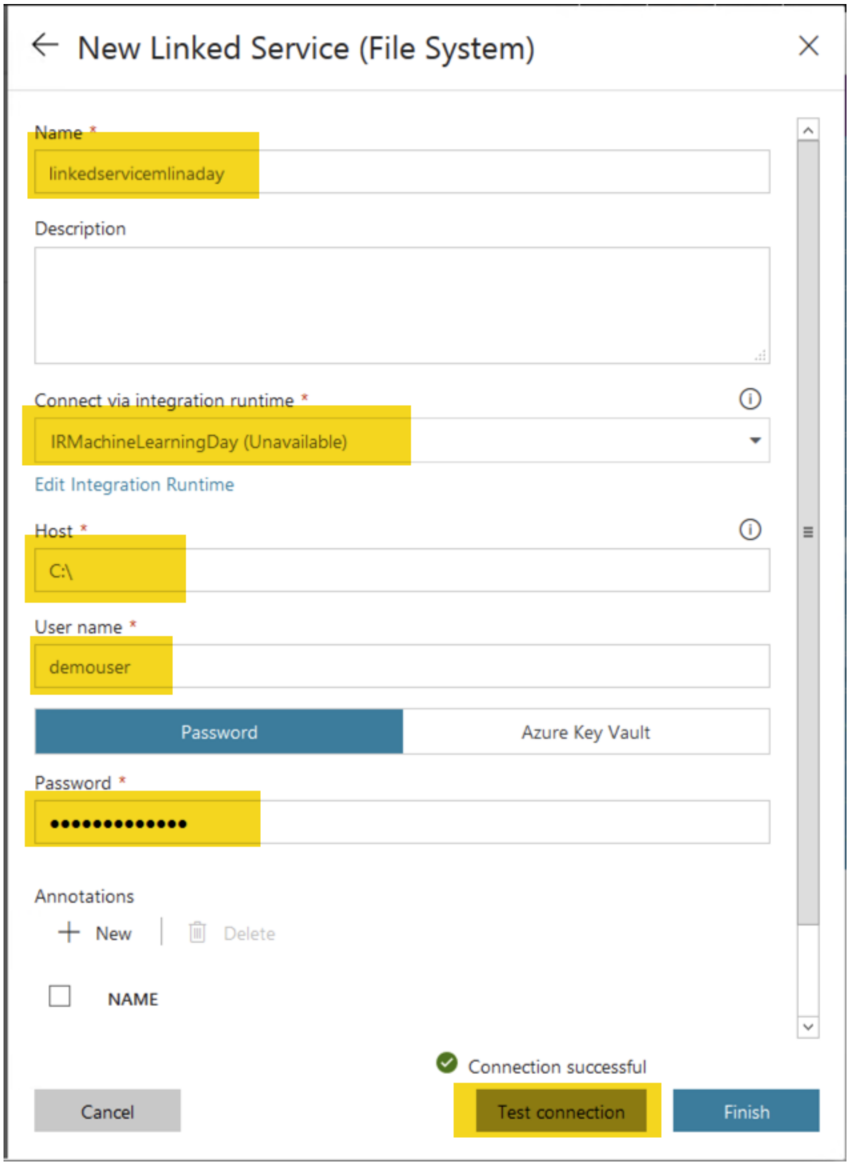
18. The Integration Runtime should begin installation. Once complete, select **Close** as seen in the following screenshot:



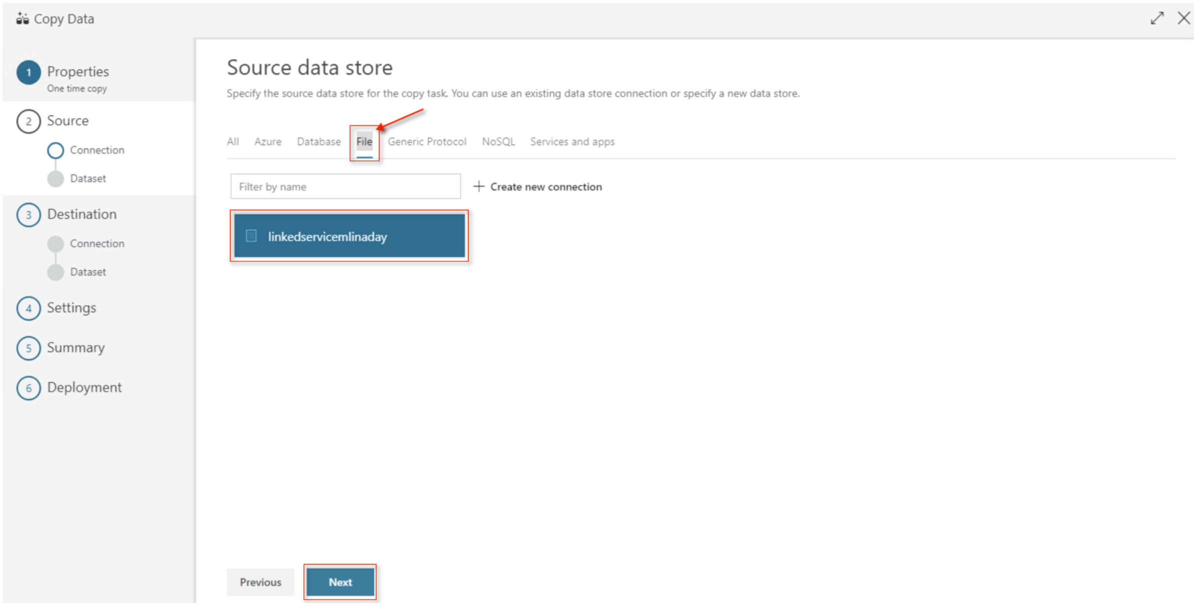
19. Select **Finish** on the **Integration Runtime Setup** as seen in the following screenshot:



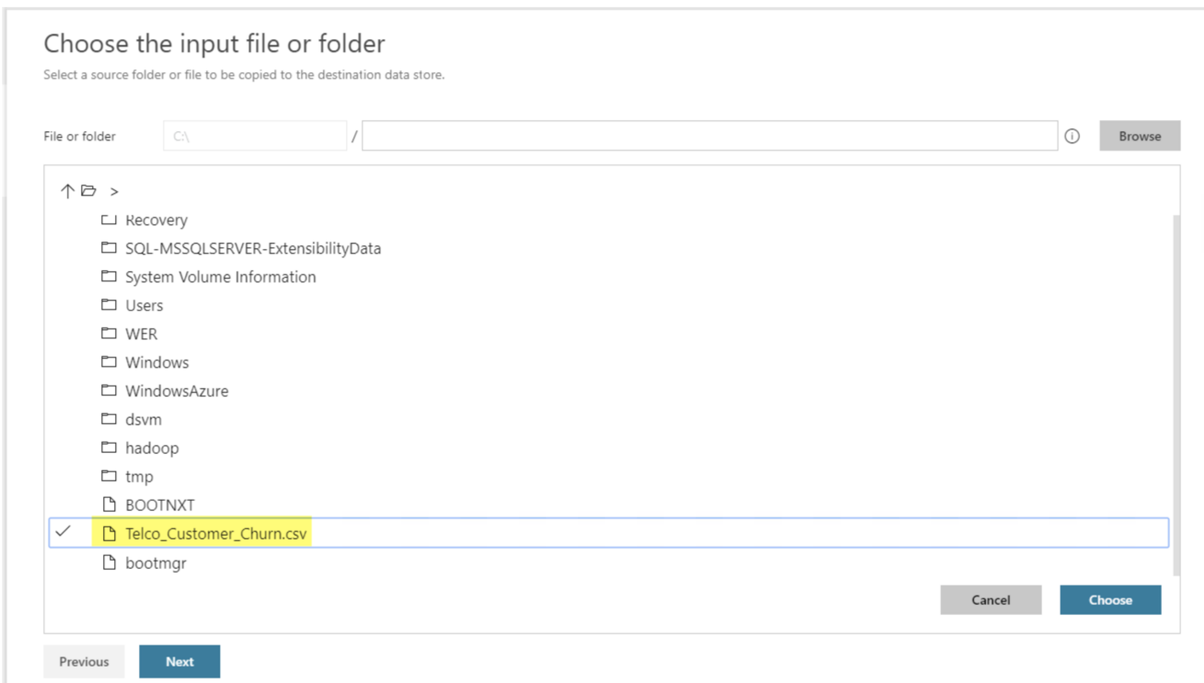
20. We can return now return to the **Linked Service (File System)** and enter our user credentials that we initially created for our Data Science Virtual Machine (DSVM) (**User name** = demouser & **Password** = #MLinaDay20!8) and enter them into our Service as well as the location of the csv file (C:\) and select **Test Connection** as seen in the following screenshot:



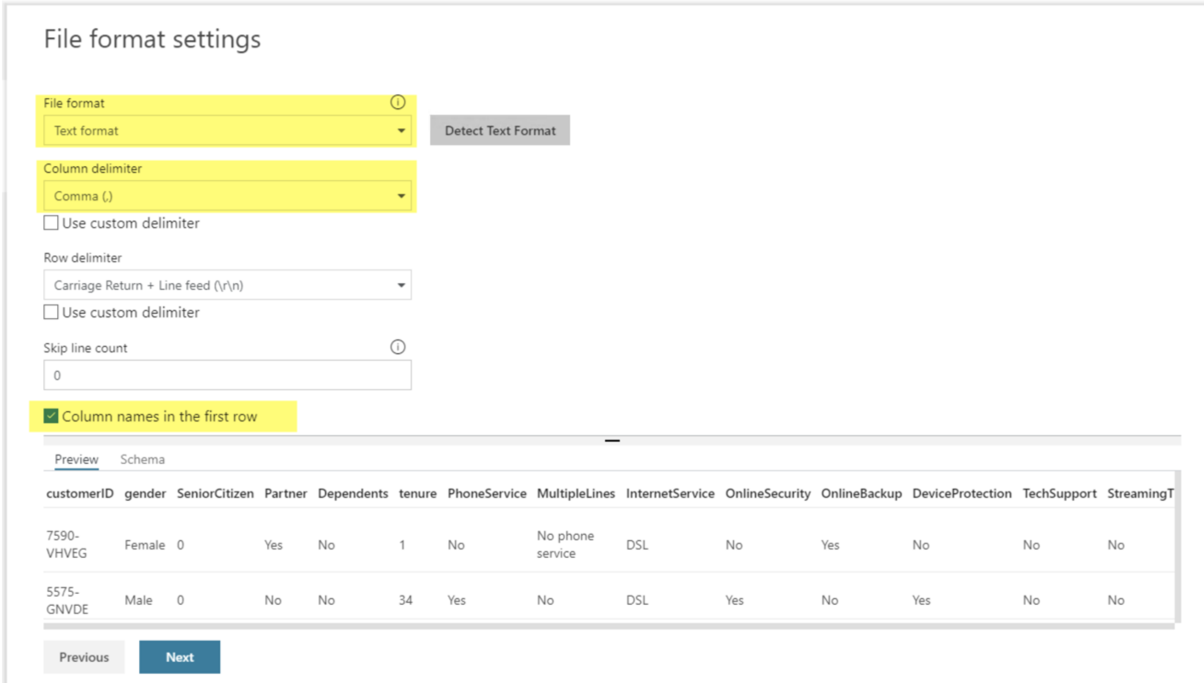
21. Hopefully we will get a **Connection Successful** message next to a green checkmark. If so, go ahead and select **Finish**.
22. Now that our Connection is set up to our virtual machine with an integration runtime, we can specify the data store type as **File** and select **Next** as seen in the following screenshot:



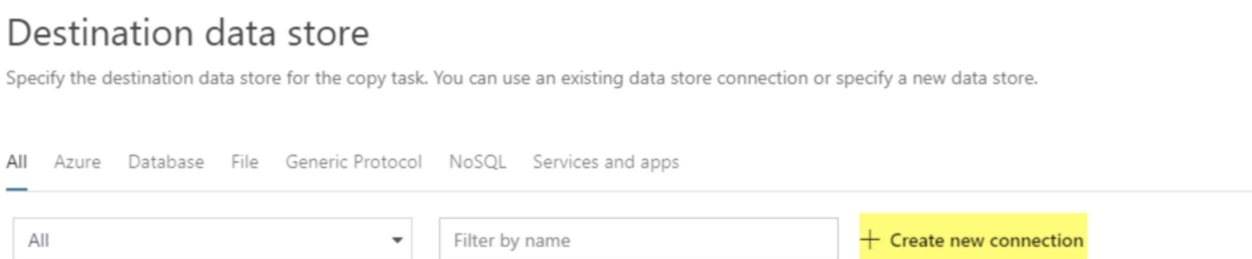
23. Choose the input file or folder as seen in the following screenshot for the **Telco_Customer_Churn.csv**:



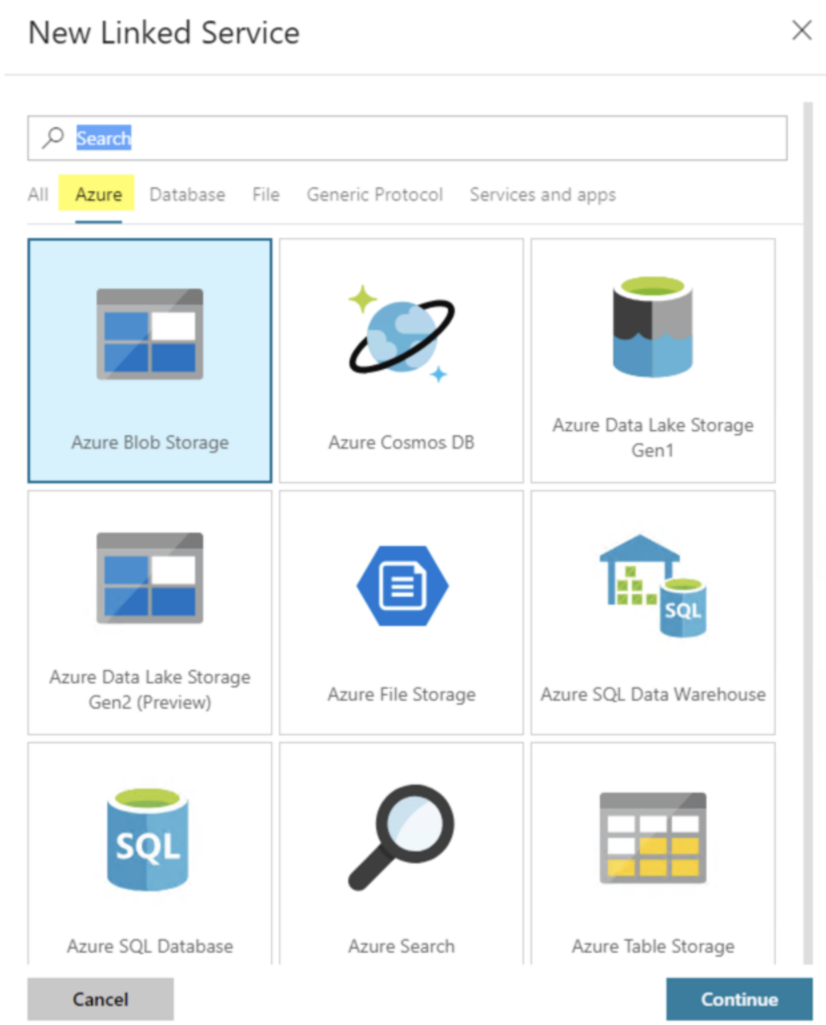
24. Specify the **File Format**, **Column Delimiter**, and select **Column Names in the first row** and select **Next** as seen in the following screenshot (we should see a preview of our dataset):



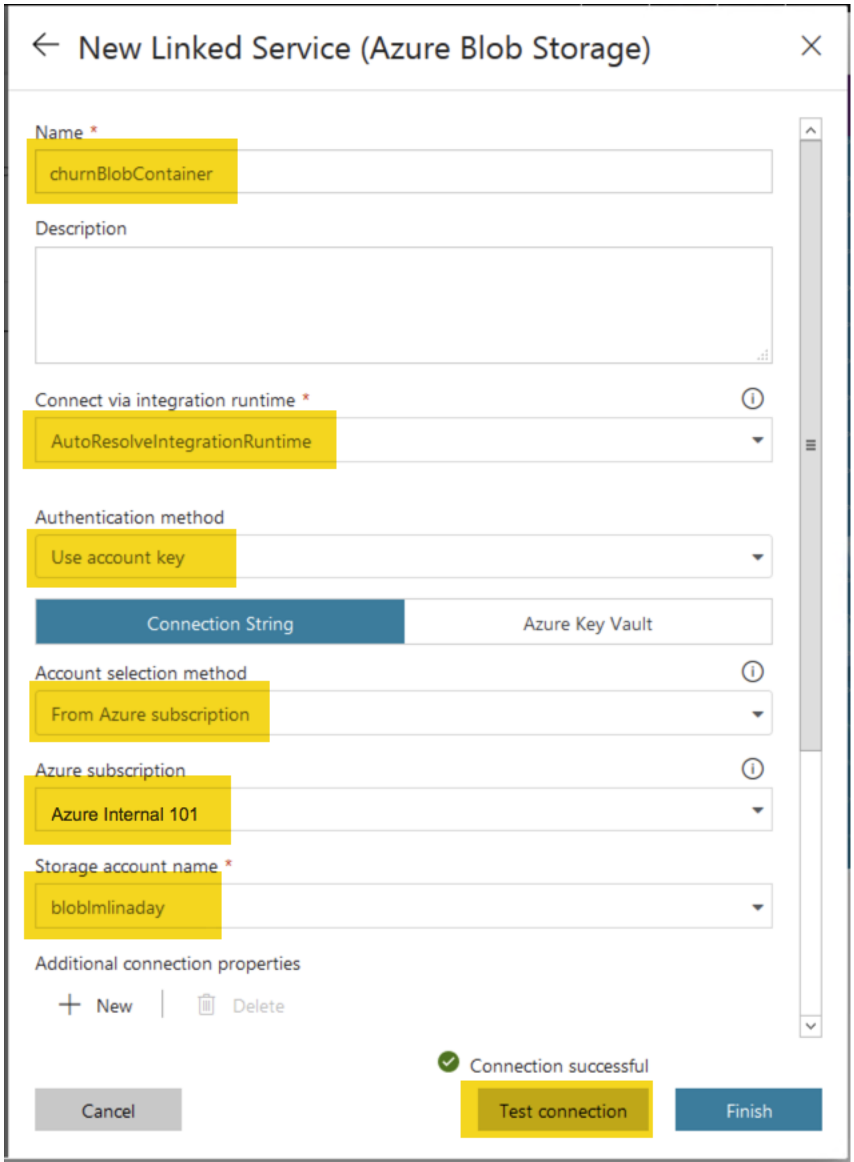
25. We are now ready to specify our **Destination data store**. We can do so by **+ Create new connection** to our Blob store container as seen in the following screenshot:



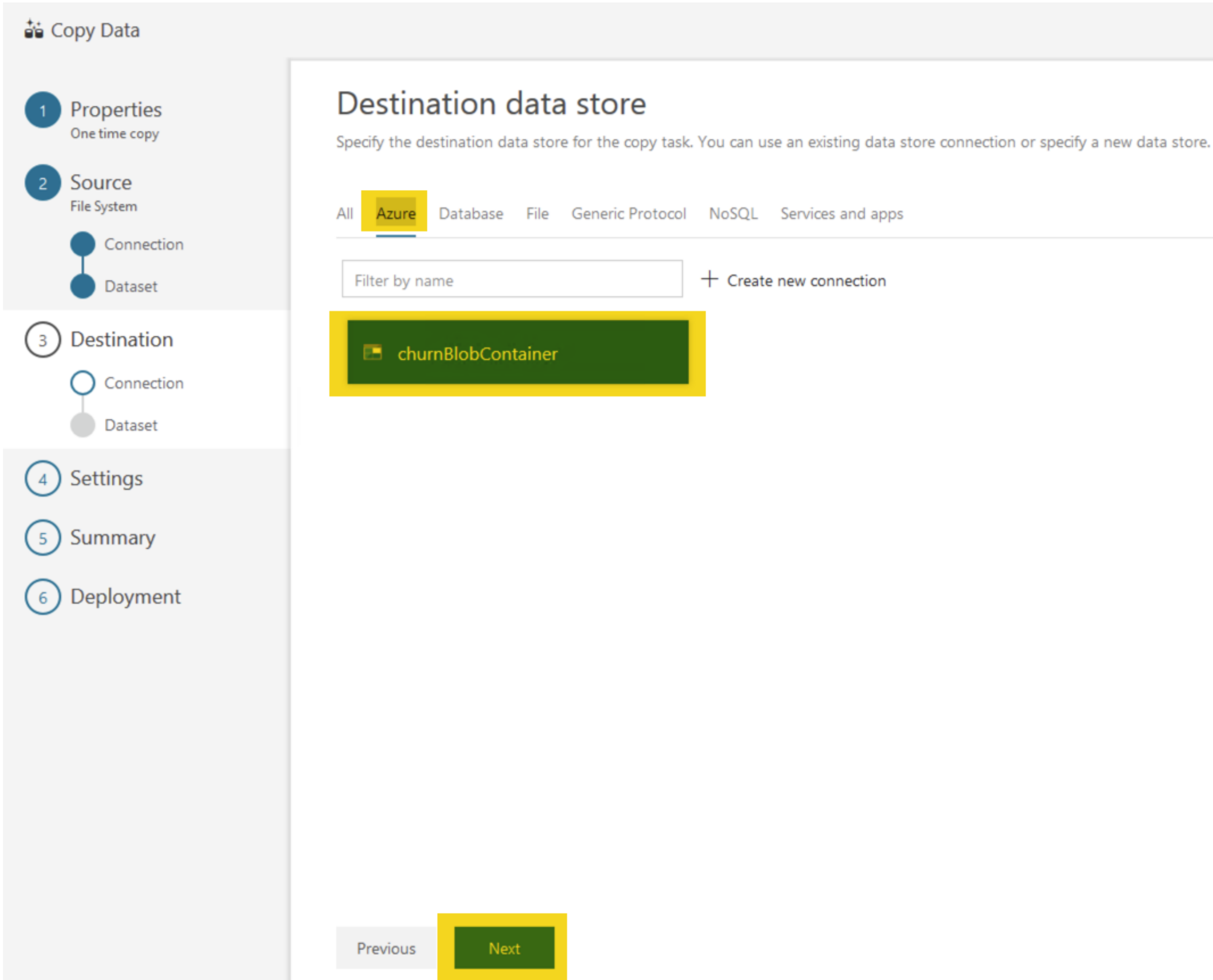
26. Search under **Azure** for **Azure Blob Store** as seen in the following screenshot:



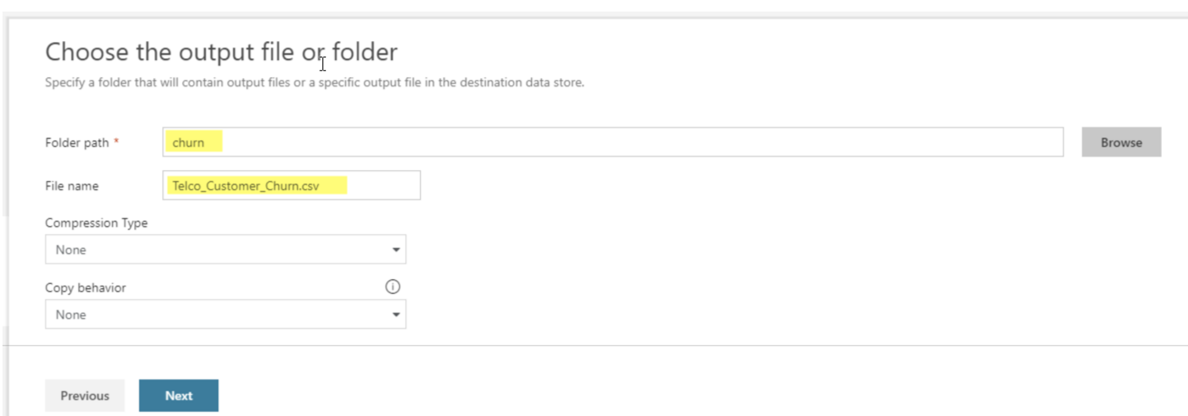
27. Specify a name for the Linked Service to Azure Blob Storage such as **churnBlobContainer**. Additionally, select your **Azure Subscription** as well as your Blob Storage Account name and **Test the Connection** as seen in the following screenshot:



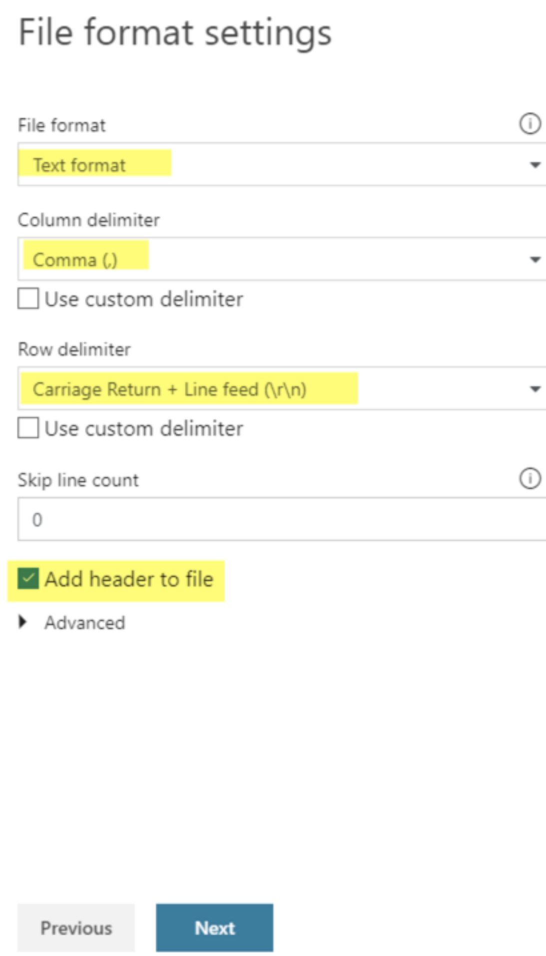
28. Once connection is successful, click on **Finish**.
29. Select the **Azure** tab as the **Destination Data Store** while also selecting the **churnBlobContainer** as the connection and then click on **Next** as seen in the following screenshot:



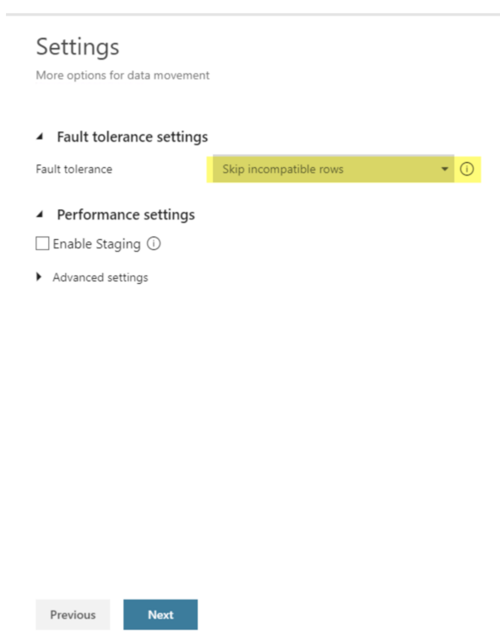
30. Specify the **folder path** which is the container (**churn**) and give a name to your file such as **Telco_Customer_Churn.csv** as seen in the following screenshot (you can leave Compression Type and Copy Behavior as None):



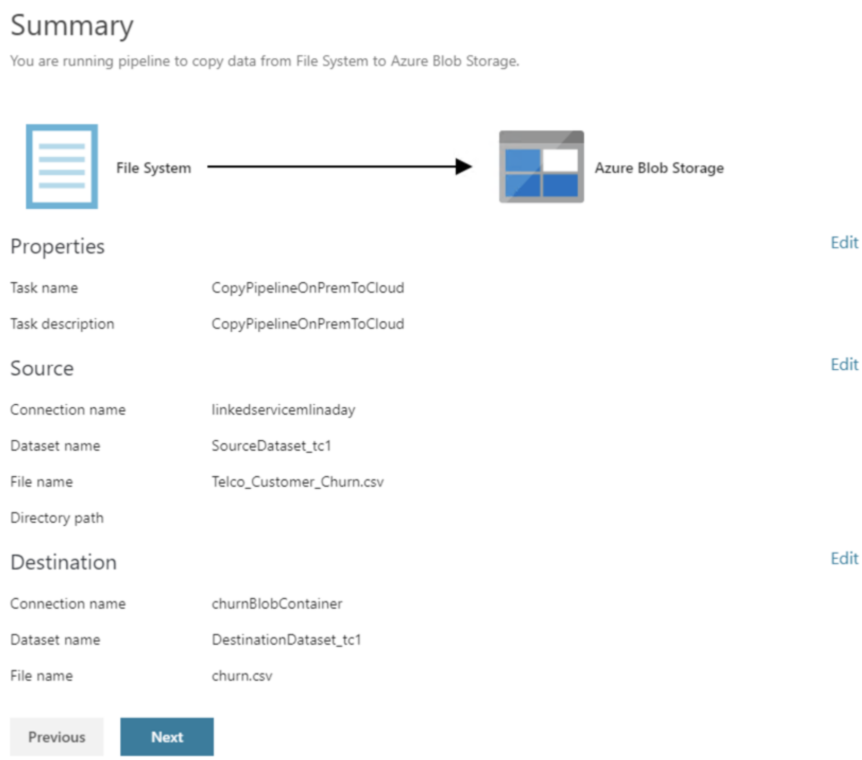
31. Specify the following **File format settings** and select **Next** as seen in the following screenshot:



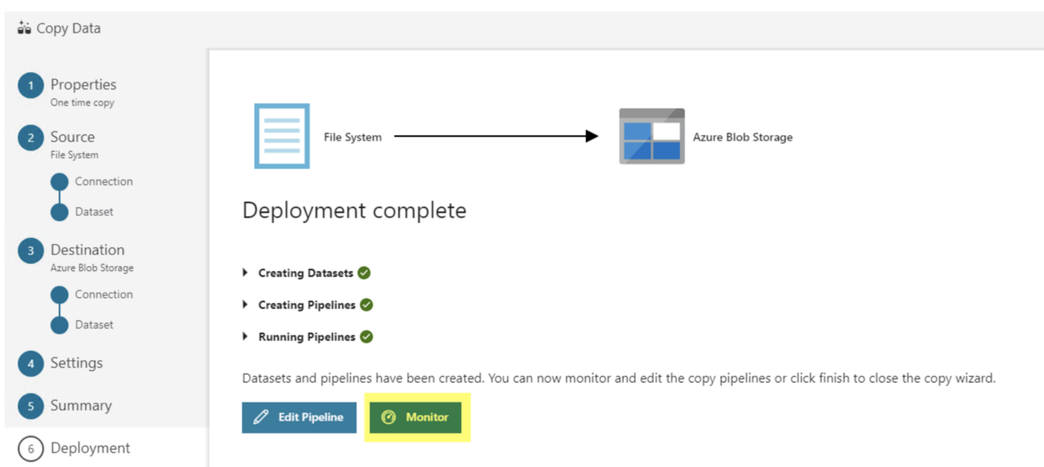
32. Set **Fault tolerance setting** to **Skip incompatible Rows** as seen in the following screenshot:



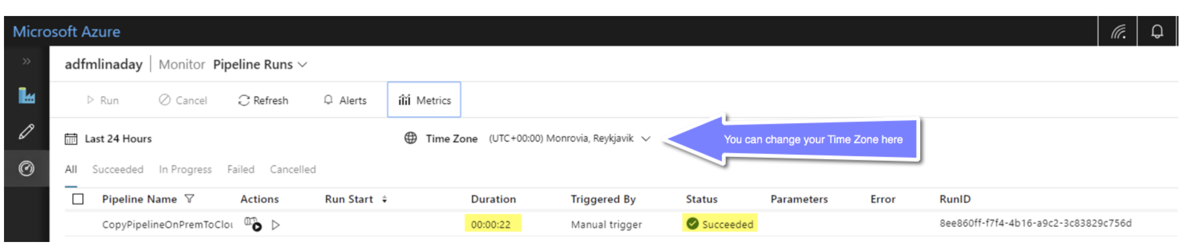
33. Review your summary pipeline to confirm all selections are accurate and select **Next** as seen in the following screenshot:



34. **Deployment** is now complete and progress can be seen by clicking on **Monitor** as seen in the following screenshot:



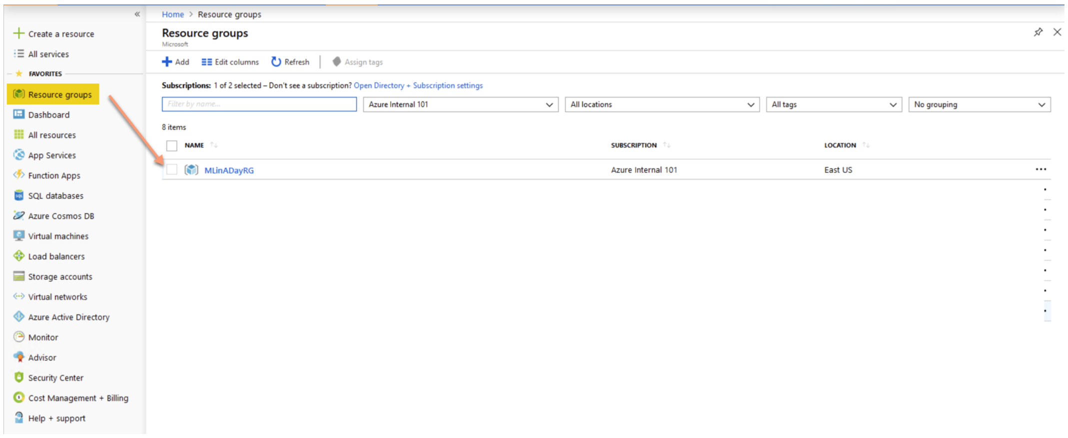
35. The **Monitor** tab will give us some summary statistics as to whether or not our deployment succeeded and how long it took as seen in the following screenshot:



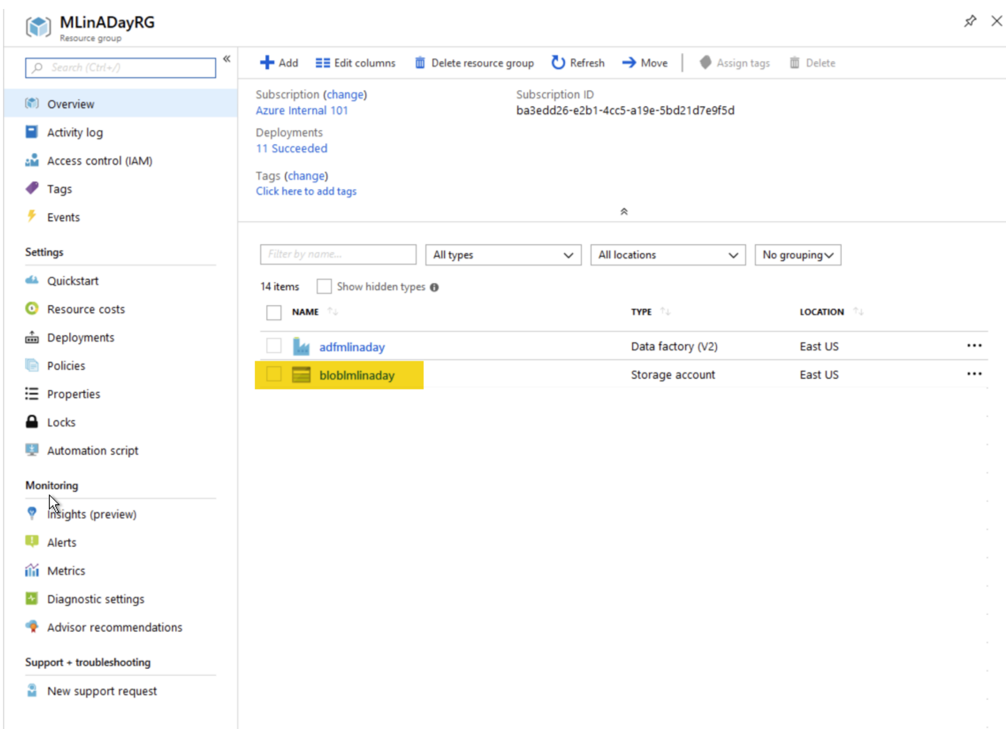
36. We can return to our blob container and confirm that our file made it there successfully. Anytime we wish to see anything we have deployed:

a. we first go to https://portal.azure.com

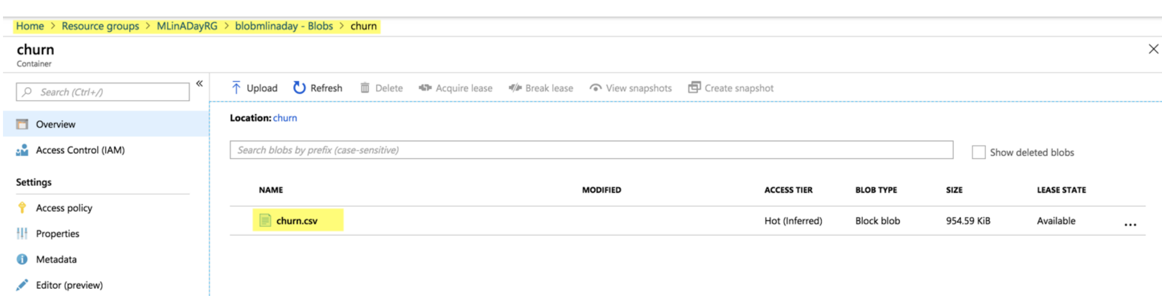
b. Then we select **Resource Groups** on the left hand side as seen in the following screenshot:



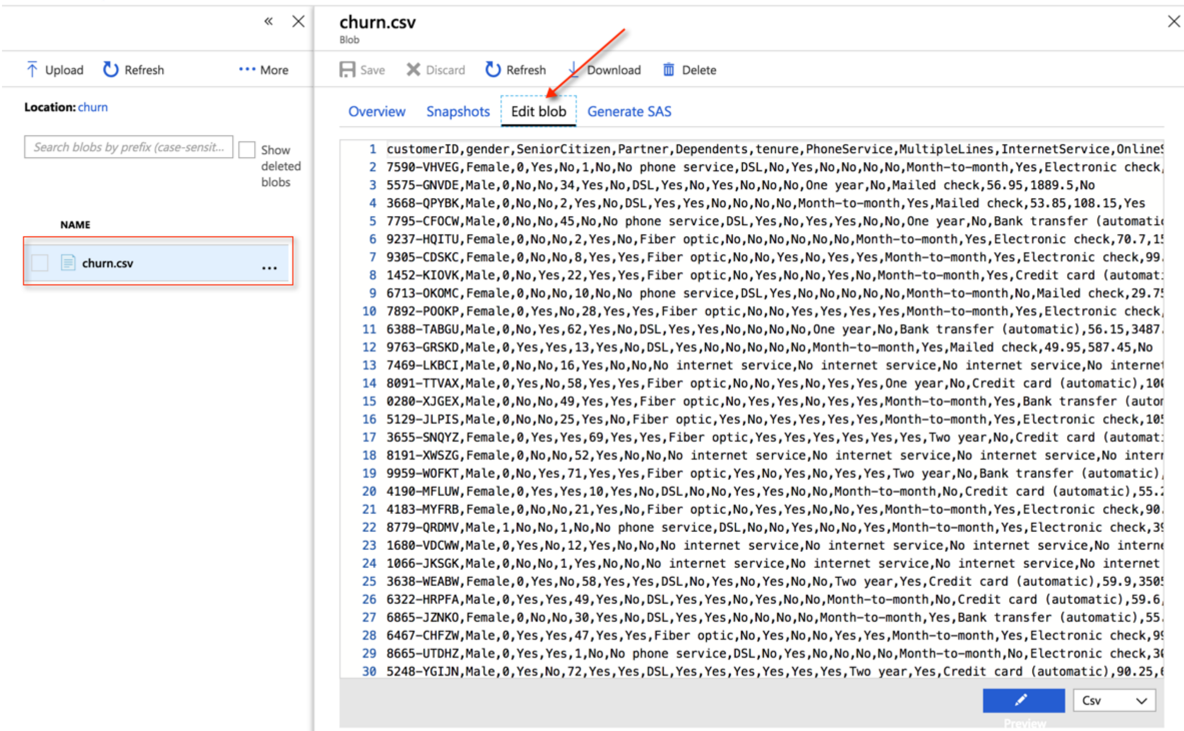
c. Finally, we can view any of the resources we have created as seen in the following screenshot:



37. We can do a final test and see that our container now holds the **churn.csv** file in our blob container as seen in the following screenshot:

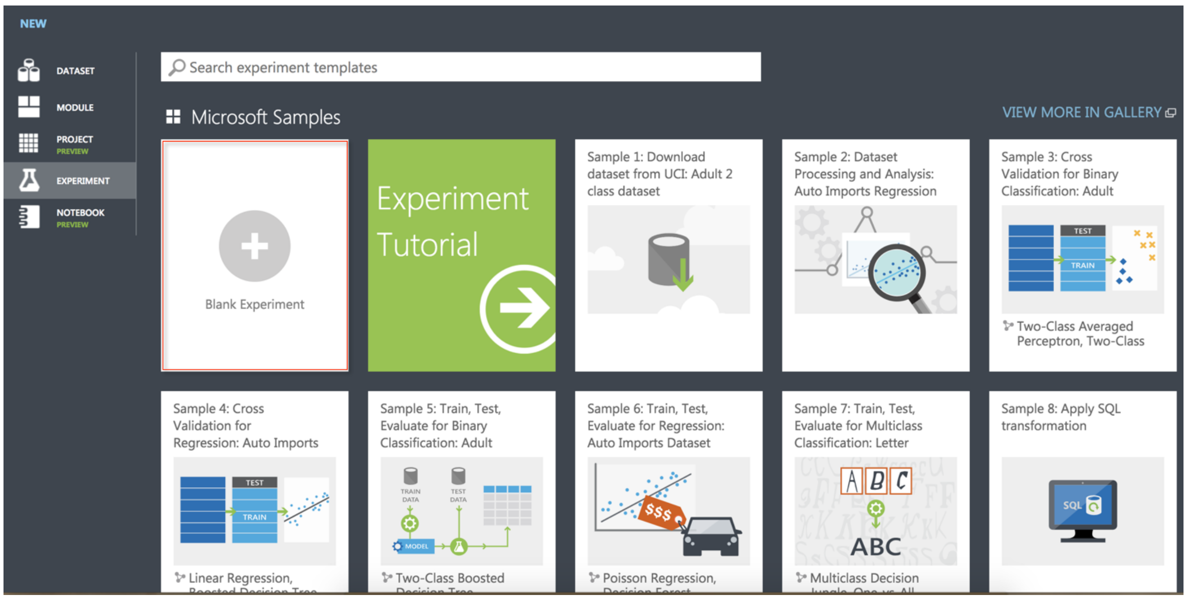


38. A simple click on the **churn.csv** file in the container and selecting **Edit blob** will reveal the contents of the file and confirm that our deployment to the cloud from on-premise was successful as seen in the following screenshot:

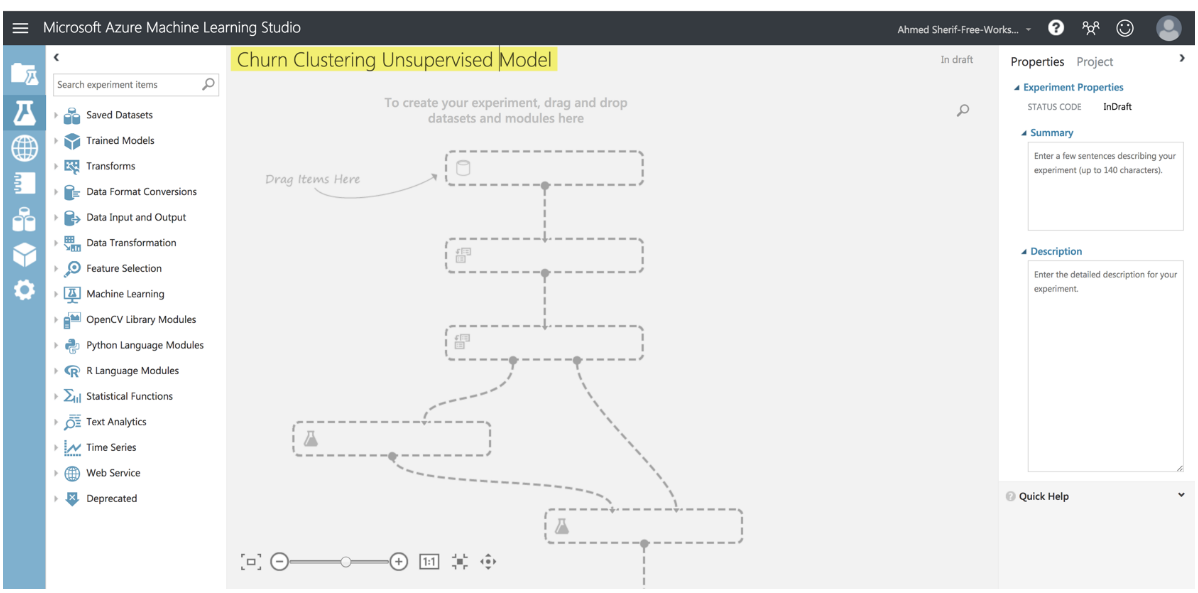


Section 4: Build a clustering unsupervised model in Azure Machine Learning Studio

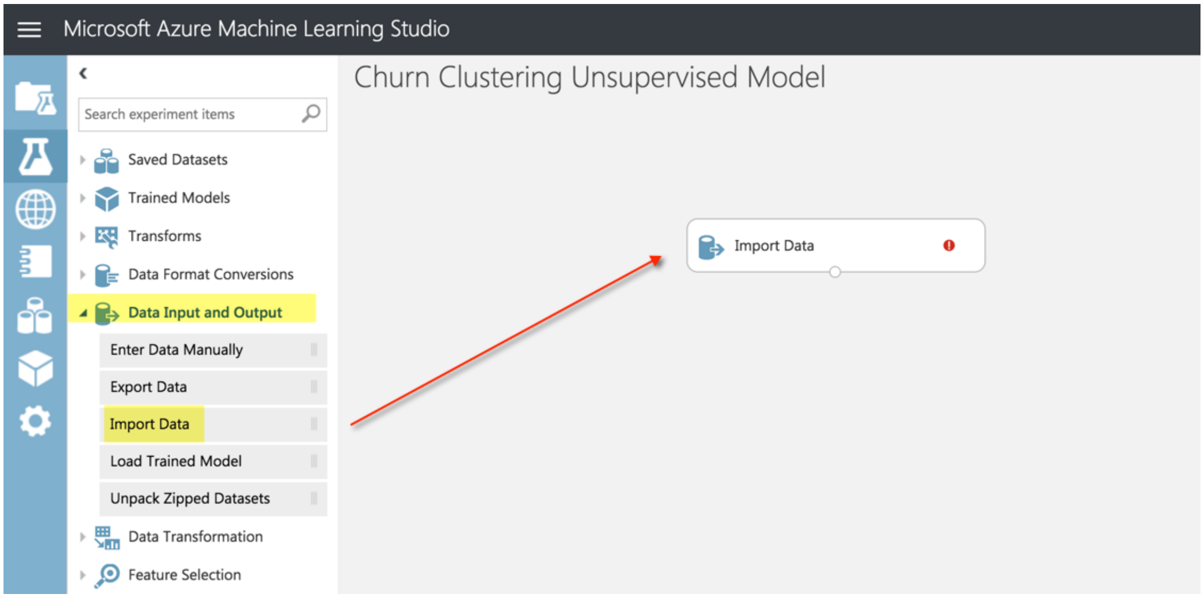
1. Clustering is the process of grouping similar entities together. The goal of this unsupervised machine learning technique is to find similarities in the fields that we have about our customers and group them into 10 distinct clusters.
2. Sign into Azure Machine Learning Studio using your Azure login Credentials in the following website: https://studio.azureml.net/
3. If you are not already directed to create a new experiment, go ahead and select the Experiments tab on the menu on the left-hand side. All current and future experiments will reside in this section.
4. Next click on **+NEW** on the button of the screen and select **Blank Experiment** as seen in the following screenshot:



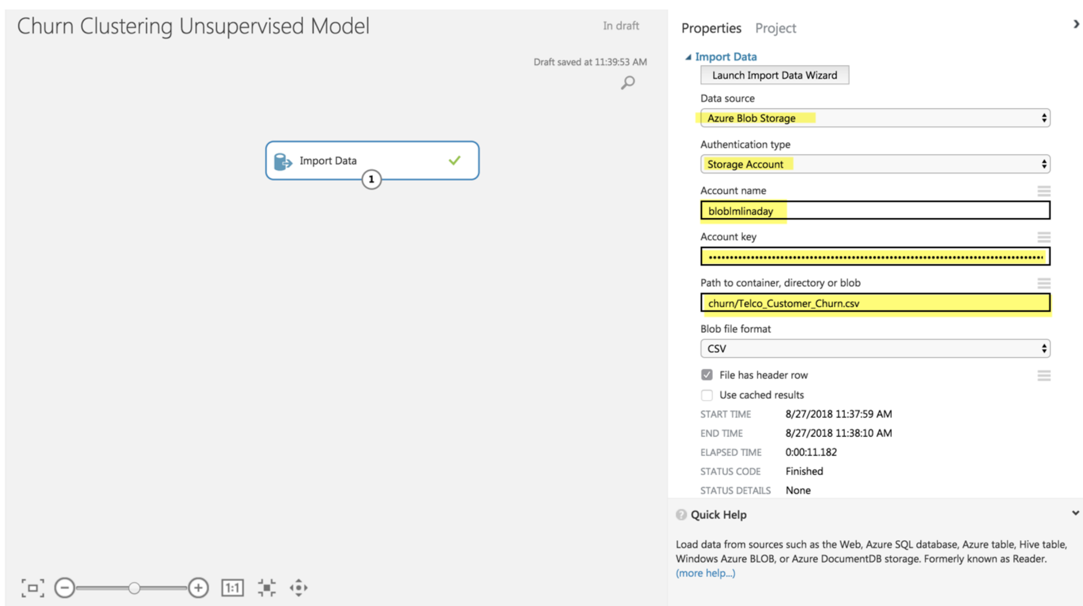
5. Click on the title of the experiment and change the name to **Churn Clustering Unsupervised Model** as seen in the following screenshot:



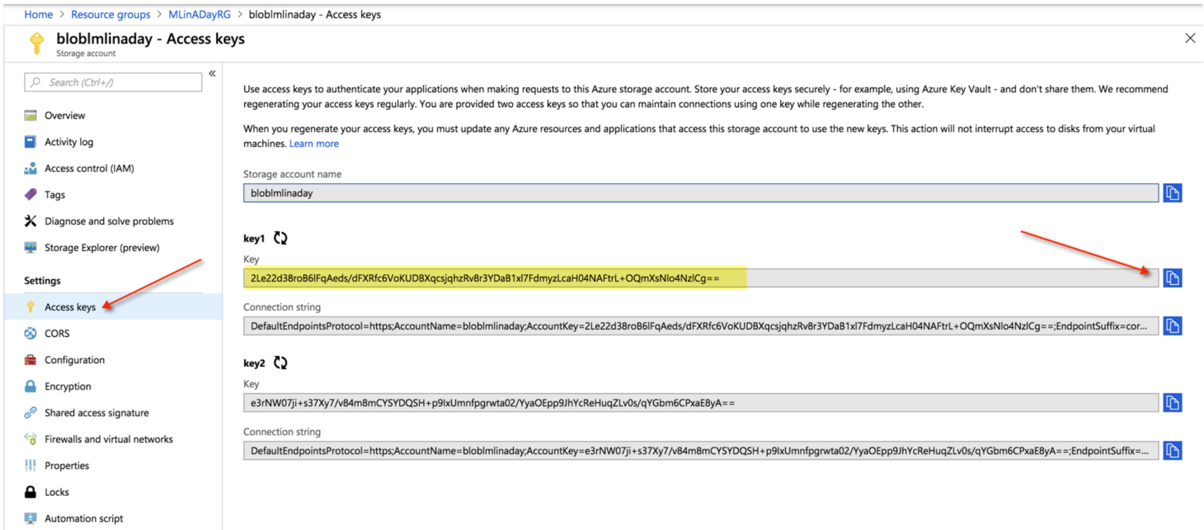
6. Expand the **Data Input and Output** category and drag the **Import Data** into the Workflow as seen in the following screenshot:



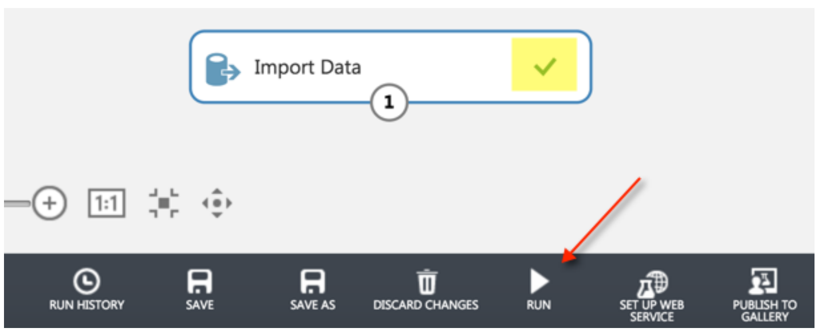
7. As you click on the **Import Data** icon in the workflow, you will have the option to configure your data to pull from the Azure Blob Container you created a few sections ago. Enter the following credentials to import your dataset into Azure ML Studio through a Blob Container connection:



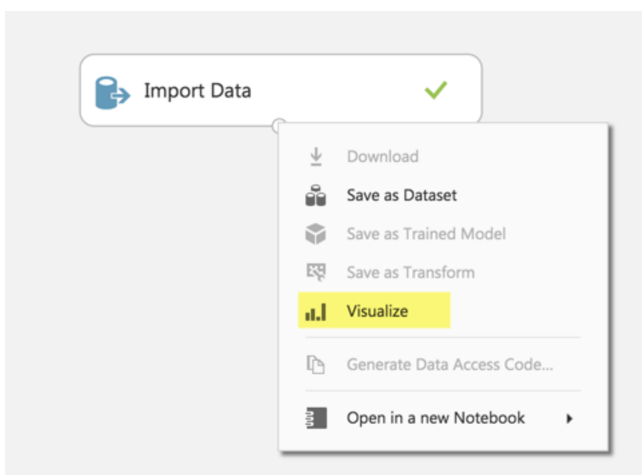
8. The Account Key is the same one that we saved a few sections ago when we created our Blob storage account and pasted it onto a Notepad editor. Once again, we will walk through the steps to obtain the **Account Key** field by going back to the Azure Portal (portal.azure.com) and copying **Key1** from the **Access Keys** section for the **blobmlinaday** account as seen in the following screenshot:



9. Back in the Azure ML Studio workflow, the credentials can be confirmed by clicking on the **RUN** icon at the bottom of the screen and seeing the green check mark next to **Import Data**.

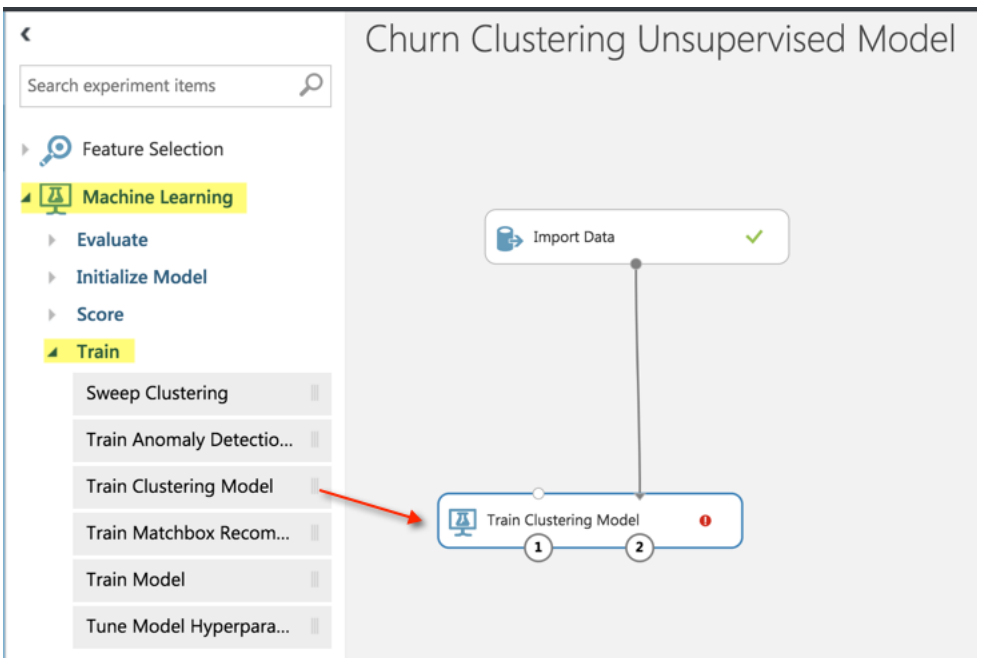


10. Once we have confirmed the connection is working, we can right-click on the number **1** on the first workflow and **Visualize** the dataset as seen in the following screenshot:

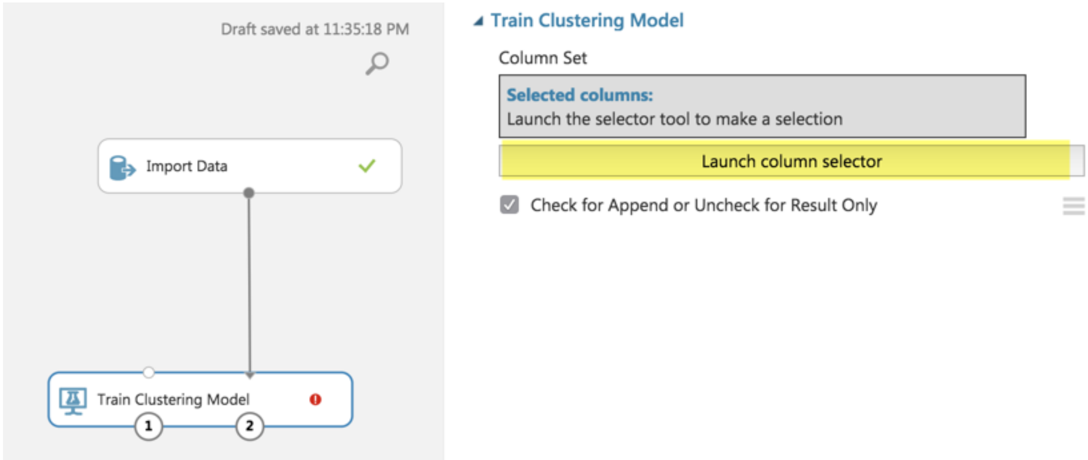


11. Take some time to click on the columns to identify the summary statistics for both the numeric fields and the categorical fields as seen in the following screenshot:

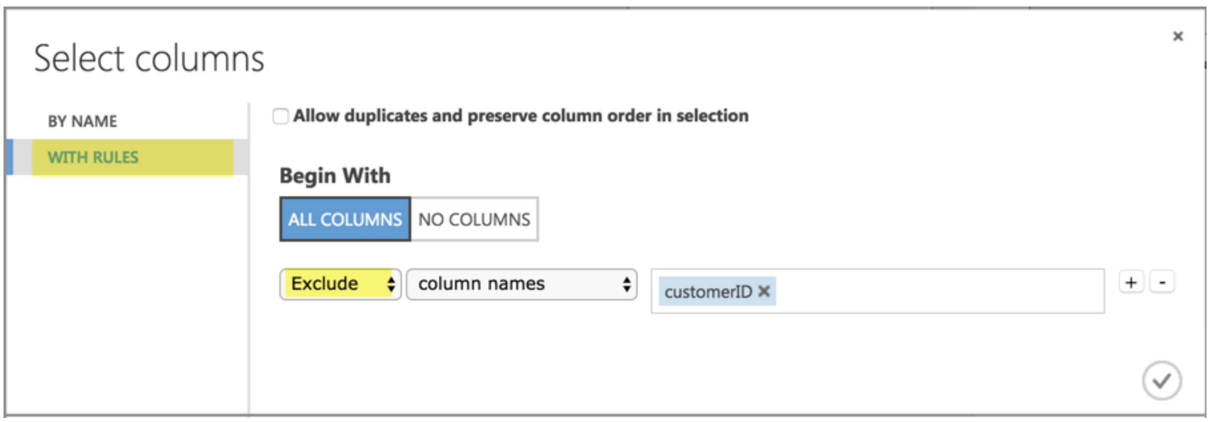
12. Import **Train Clustering Model** from the **Machine Learning** Node and connect it to the **Import Data** module as seen in the following screenshot:



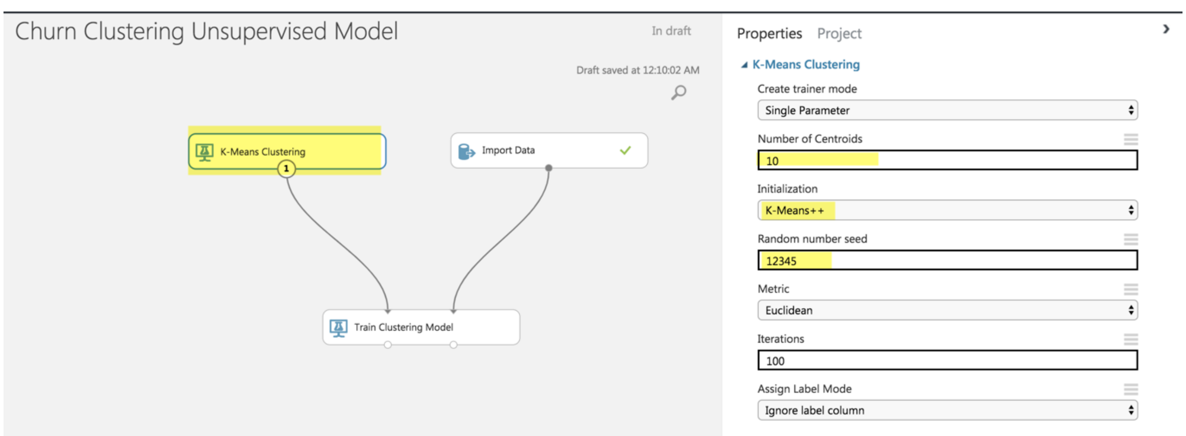
13. Next, click on the **Train Clustering Model** module, **Launch Column Selector**, connect the node back to the **Import Data** node, and configure the columns to be selected as seen in the following screenshot:



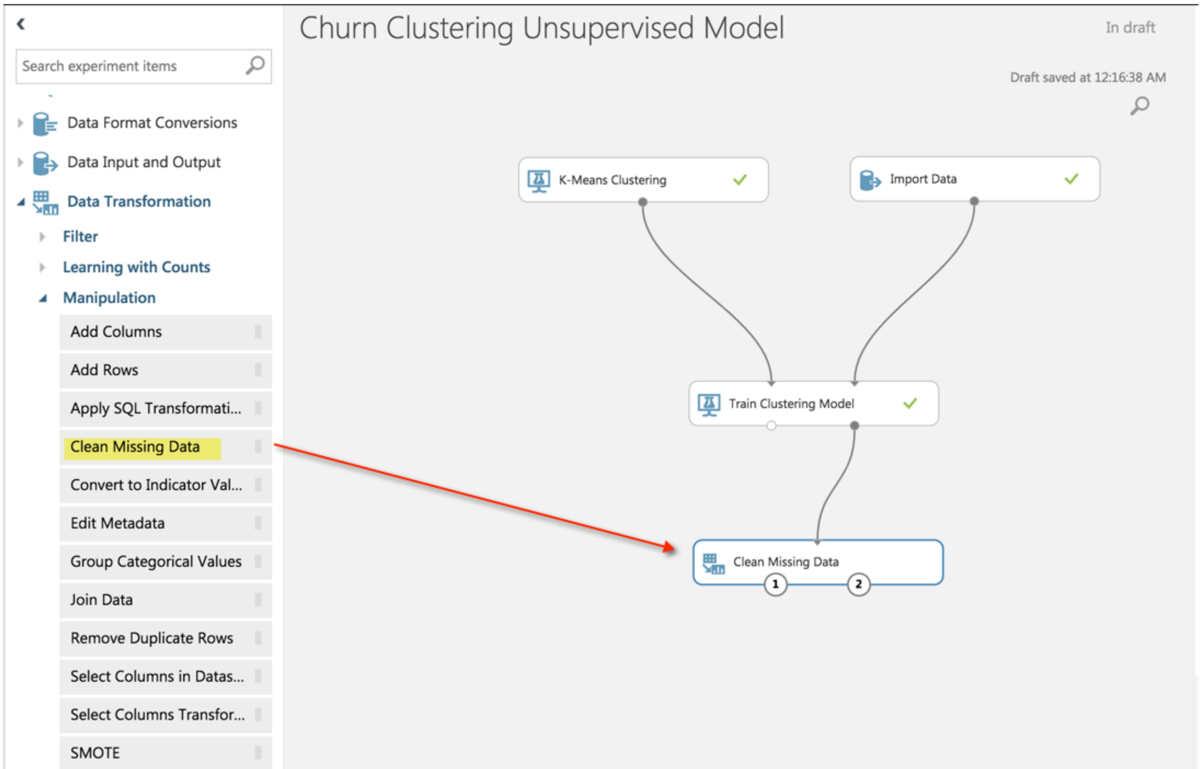
14. Next, you will need to configure the **clustering model** and **Select All columns** from the dataset except for the column called **customerID** as seen in the following screenshot:



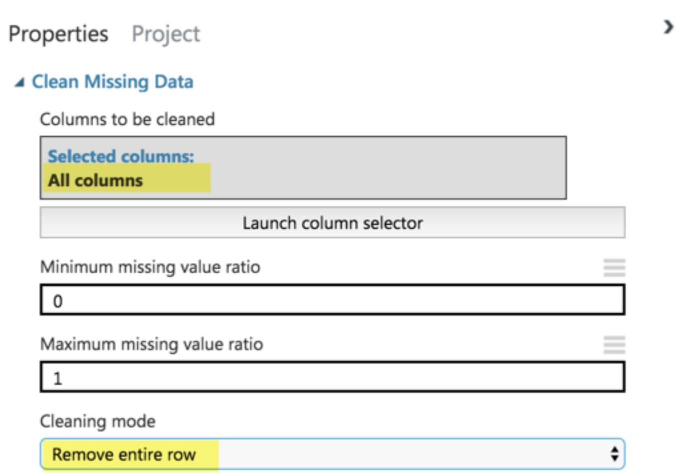
15. Next, we will attach the **K-Means Clustering** model to the **Train Clustering Model** and assign the following parameters to the clustering model, as seen in the following screenshot:



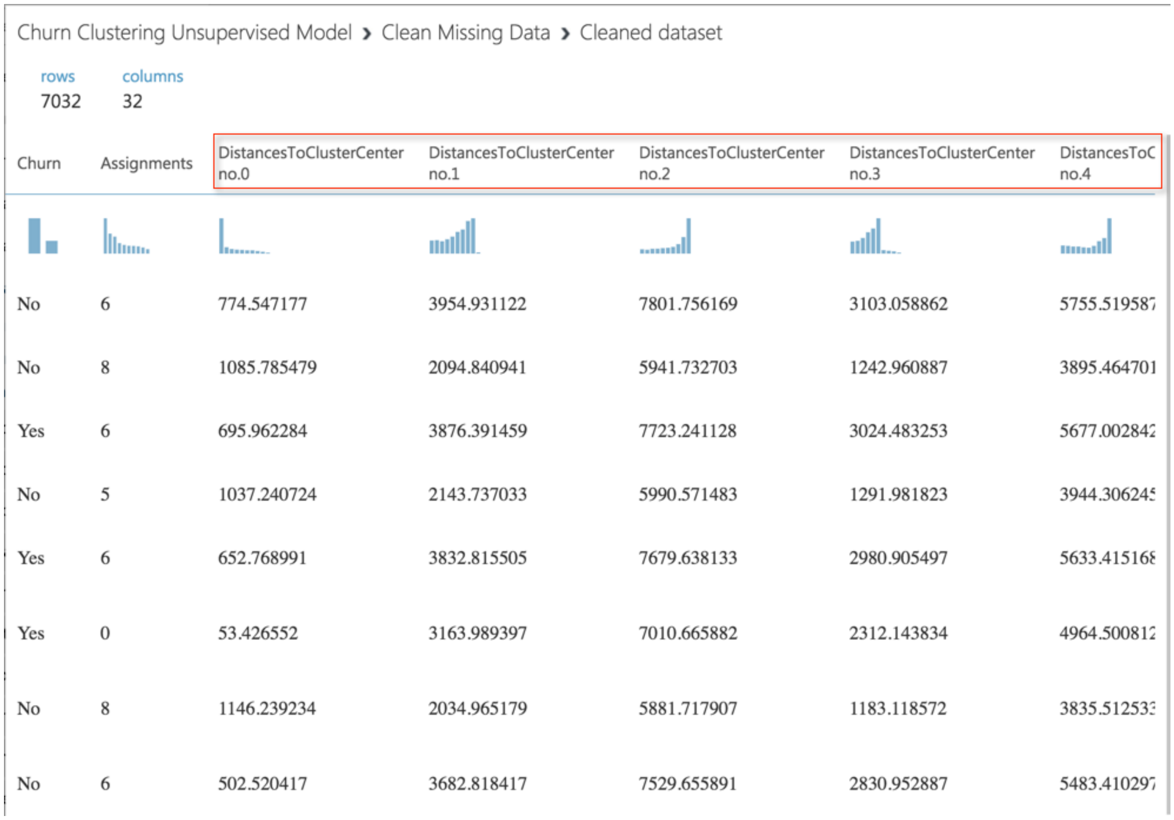
16. Next, we will do some data cleanup and remove any rows that have missing values by pulling in the **Clean Missing Data** module and connecting it to the **Train Clustering Model** as seen in the following screenshot:



17. We can configure the **Clean Missing Data** module by clicking on it and selecting the following configuration to remove any rows if they meet the following criteria:

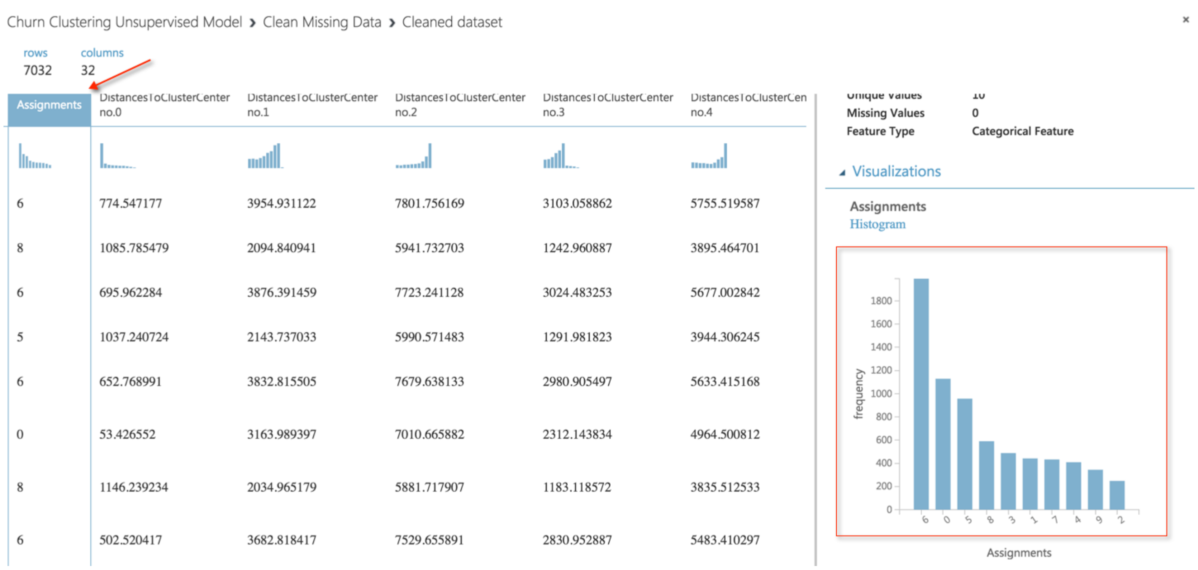


18. The clustering outcome produces several columns that are not necessary for our purposes for this workshop as can be seen by visualizing the **Clean Missing Data** module as seen in the following screenshot:

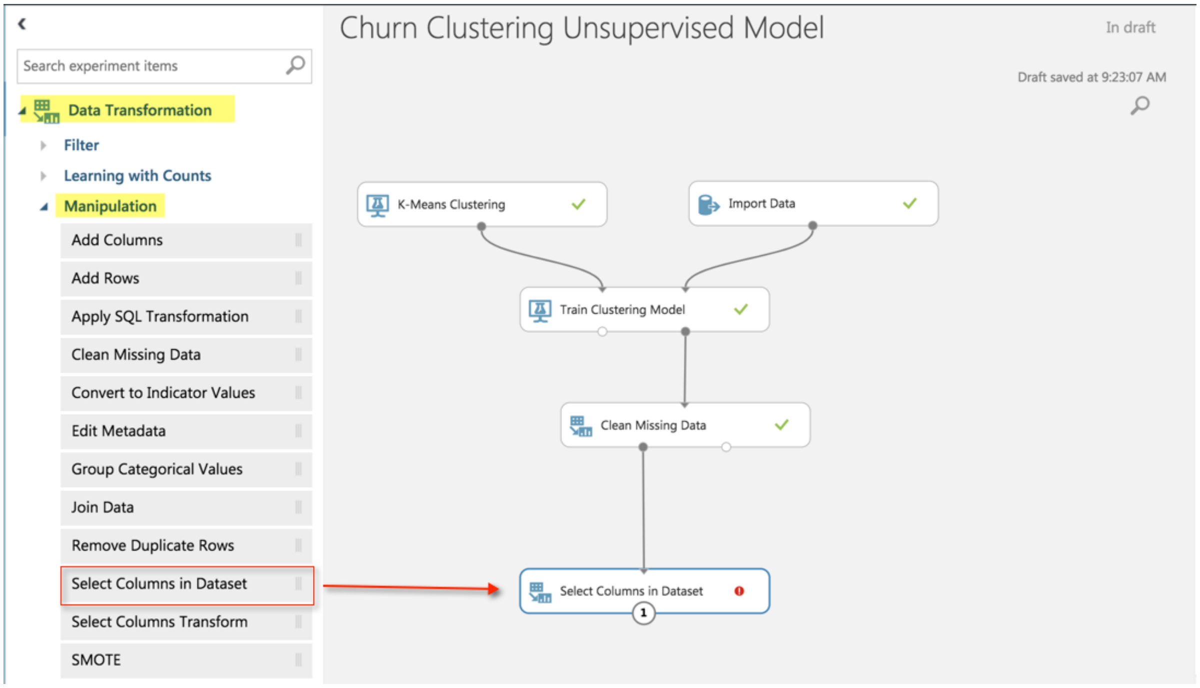


Churn	Assignments	DistancesToClusterCenter no.0	DistancesToClusterCenter no.1	DistancesToClusterCenter no.2	DistancesToClusterCenter no.3	DistancesToC no.4
No	6	774.547177	3954.931122	7801.756169	3103.058862	5755.51958
No	8	1085.785479	2094.840941	5941.732703	1242.960887	3895.464701
Yes	6	695.962284	3876.391459	7723.241128	3024.483253	5677.00284
No	5	1037.240724	2143.737033	5990.571483	1291.981823	3944.30624
Yes	6	652.768991	3832.815505	7679.638133	2980.905497	5633.41516
Yes	0	53.426552	3163.989397	7010.665882	2312.143834	4964.50081
No	8	1146.239234	2034.965179	5881.717907	1183.118572	3835.51253
No	6	502.520417	3682.818417	7529.655891	2830.952887	5483.41029

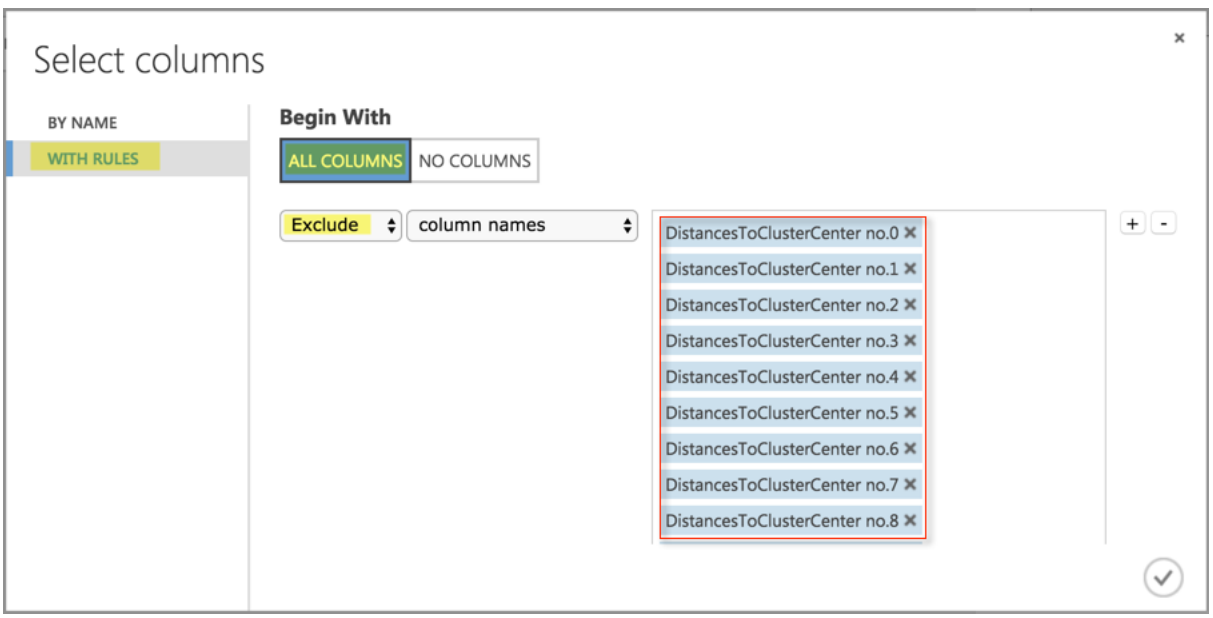
19. We are truly only interested in the assignments, which is grouping each customer into a category of **0** through **9** as seen when we visualize a histogram of the unique values as seen in the following screenshot:



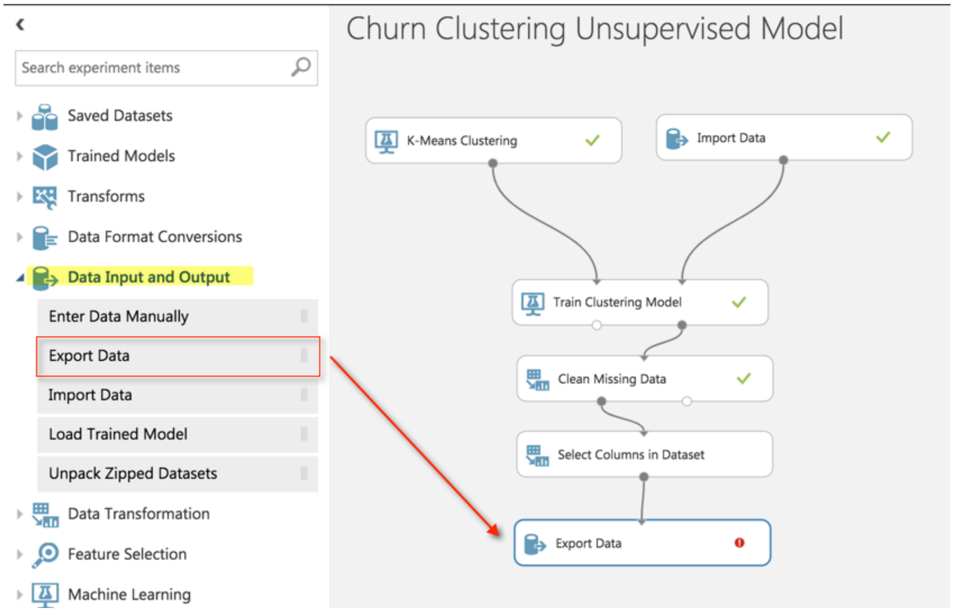
20. All other columns that start with **DistancesToClusterCenter** are not relevant for our purposes and can be removed. The easiest way to do this is to drag into our workflow **Select Columns in Dataset** as seen in the following screenshot:



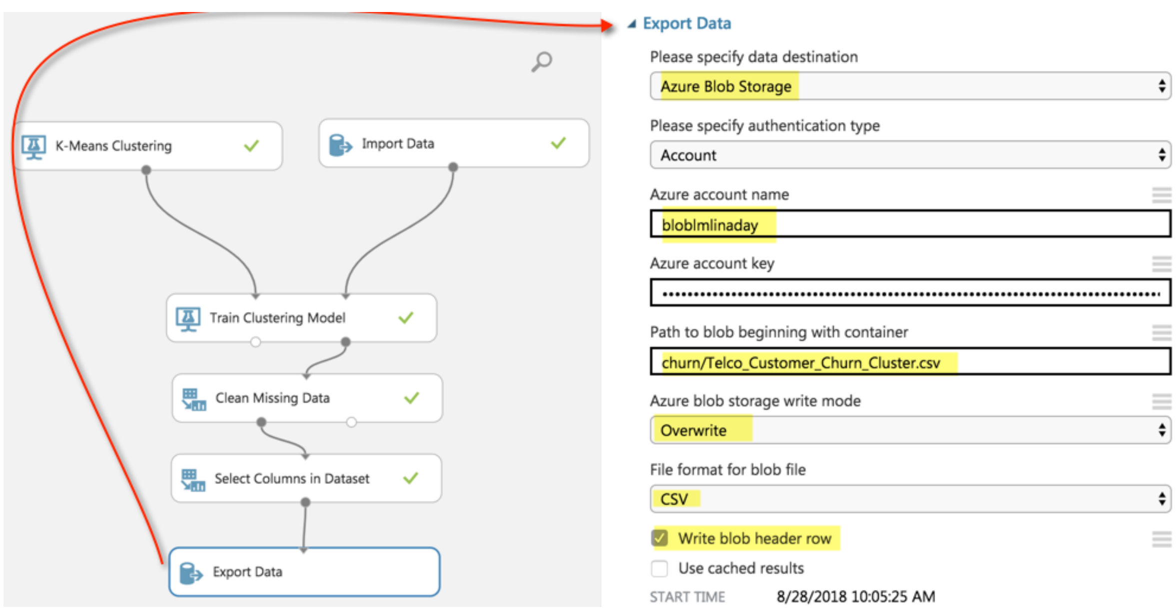
21. We can then configure the module for **Select Columns in Dataset** by **Launching Column Selector**, select WITH RULES, and exclude column names as seen in the following screenshot:



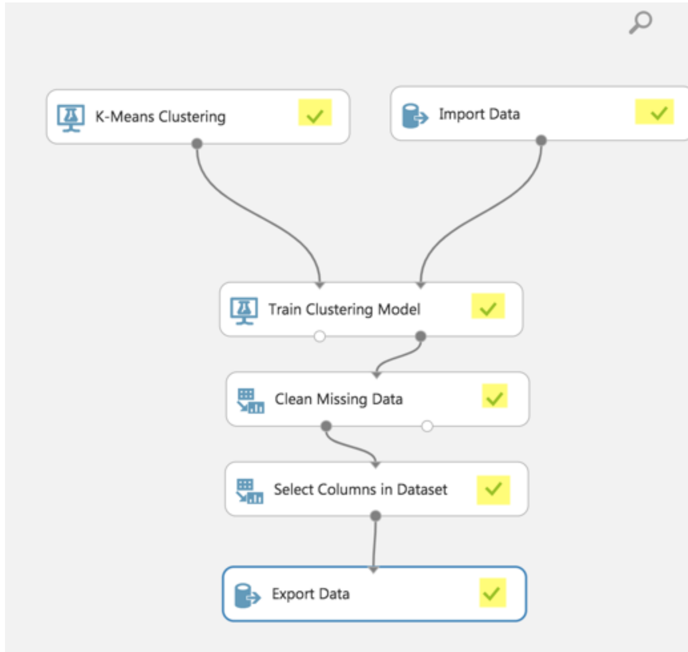
22. We are finally ready to send out revised dataset with only the columns that we need back to blob storage using the Export Data module as seen in the following screenshot:



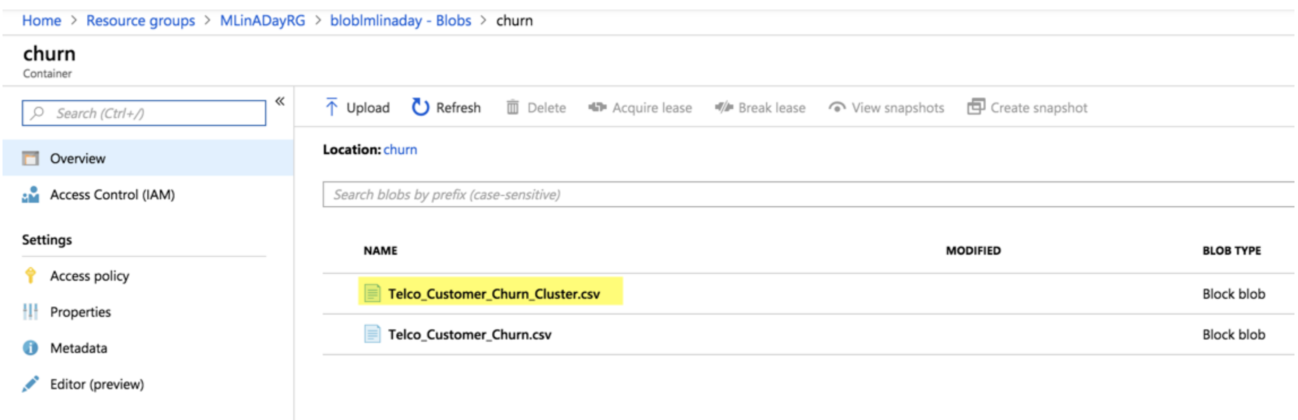
23. Just as we did with importing the data into AML Studio from Blob, we will need to configure the connection to export the new dataset, **Telco_Customer_Churn_Cluster.csv**, back to the same blob container, **churn**, using the following configuration as seen in the following screenshot *(please note that you should continue to use the same **Key1** from Blob storage that was used earlier on in the section when importing the data set from Blob)*:



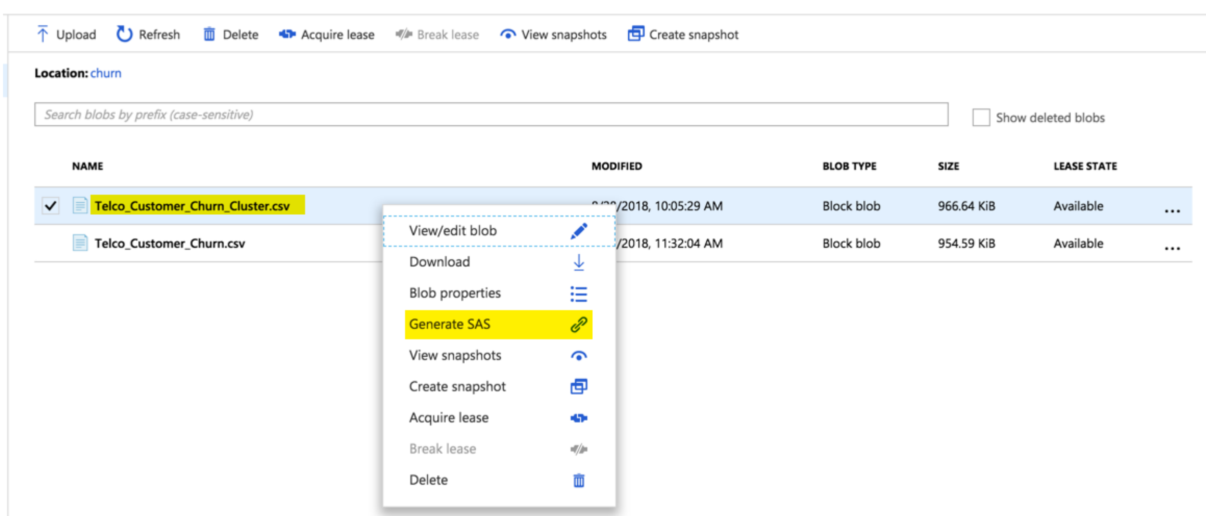
24. Once we are done, we can execute our workflow one last time by clicking on the **RUN** button. Once completed, we should hopefully see all green checkmarks next to each step of our workflow as seen in the following screenshot:



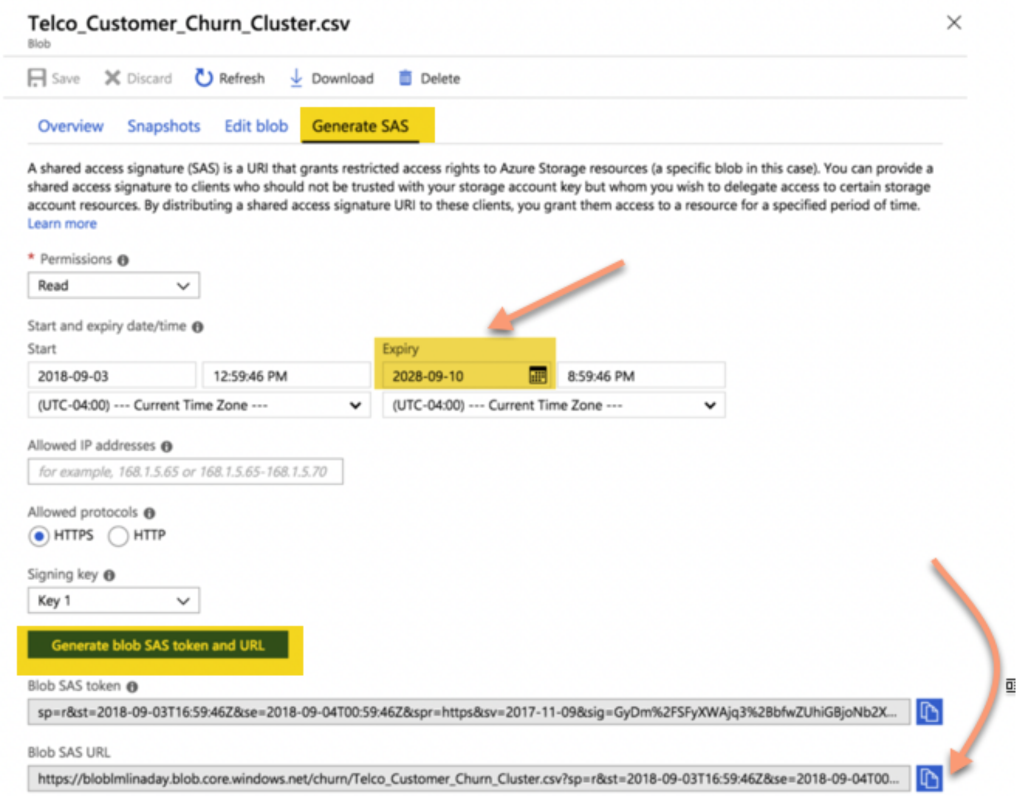
25. We can revisit our Blob container in the Azure Portal and view our newly added dataset alongside our existing dataset as seen in the following screenshot:



26. Finally, we will right-click on the csv file and obtain our **Generate SAS** as seen in the following screenshot:



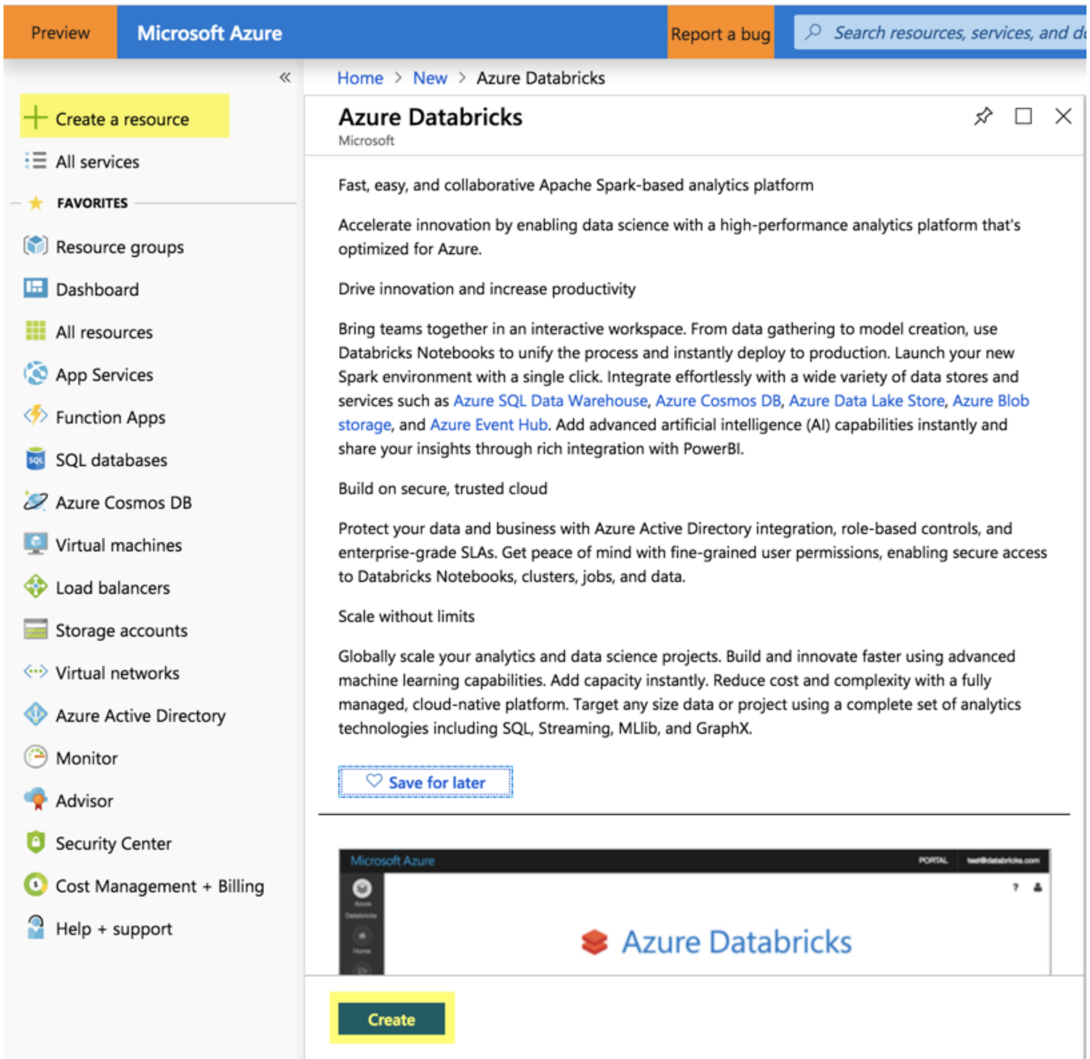
27. A **shared access signature (SAS)** is a URL that grants restricted access rights to Azure Storage resources (a specific blob in this case). You can provide a shared access signature to clients who should not be trusted with your storage account key but whom you wish to delegate access to certain storage account resources. By distributing a shared access signature URI to these clients, you grant them access to a resource for a specified period of time. Select Generate blob SAS token and URL as seen in the following screenshot:



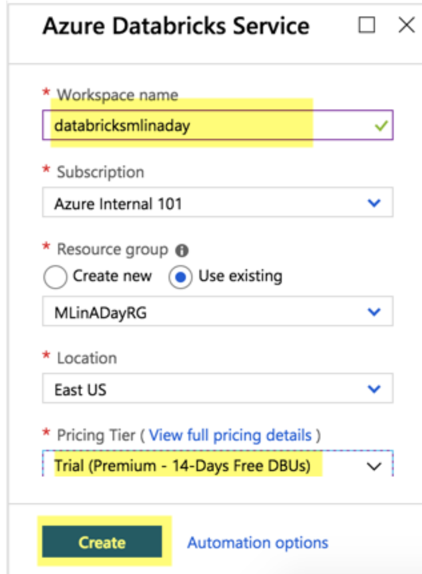
28. Keep a copy of the SAS URL as it will come in handy in the next section when we are reading the file from Blob to Azure Databricks. Additionally, make sure to set expiry date sometime in the future so that it does not expire while you are working on the workshop:

Section 5: Set up a Spark Cluster on Databricks and connect to Azure Blob Storage Account

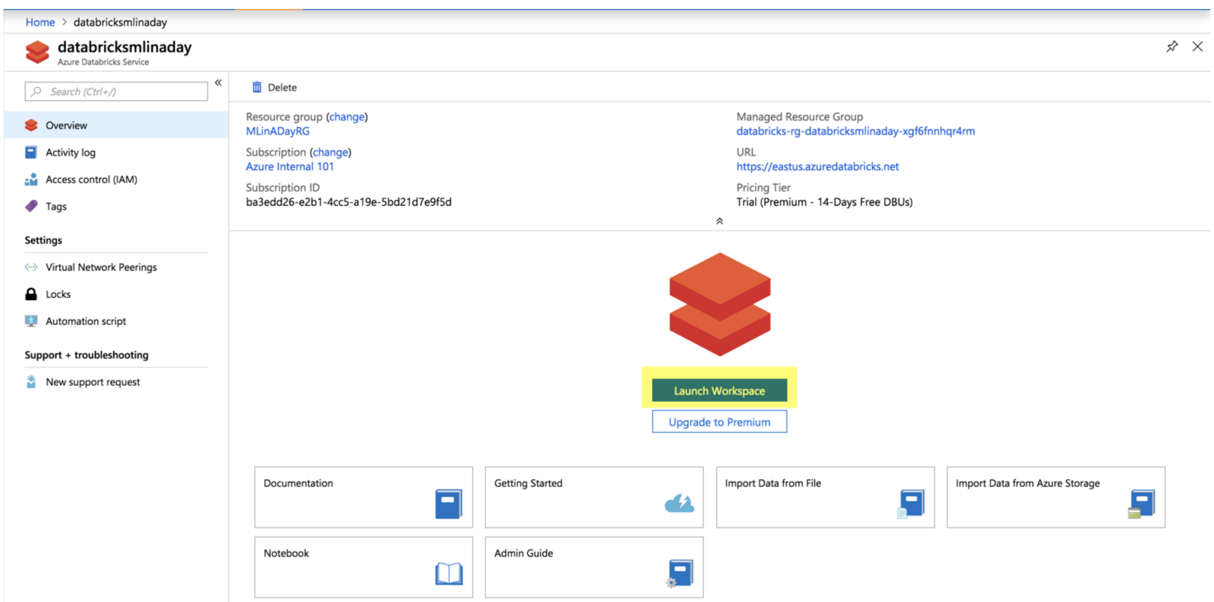
1. Return to the Azure portal (https://portal.azure.com) and provision an **Azure Databricks** workspace by selecting Azure Databricks from the **marketplace** as seen in the following screenshot:



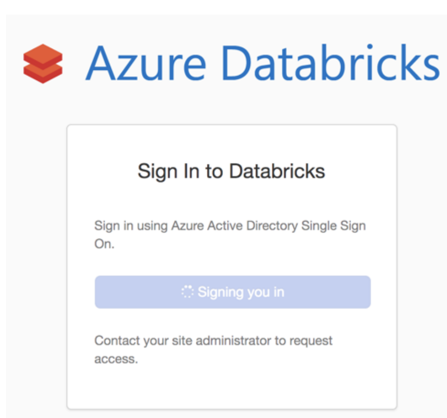
2. Enter the **Workspace** specifications as seen in the following screenshot:



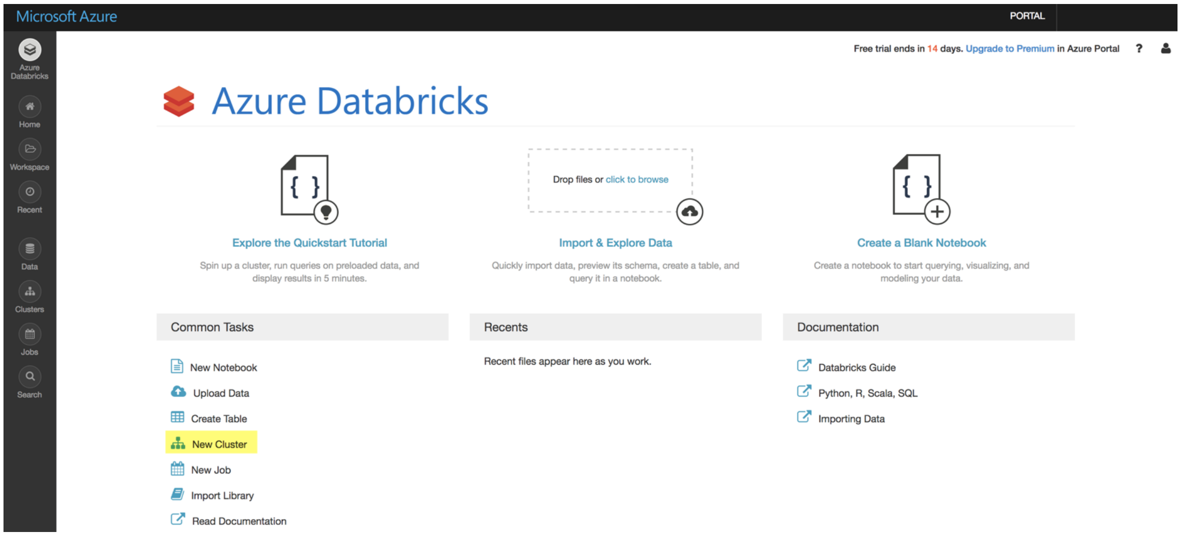
3. Once it has been provisioned, you can go to the resource and **launch** the **workspace** as seen in the following screenshot:



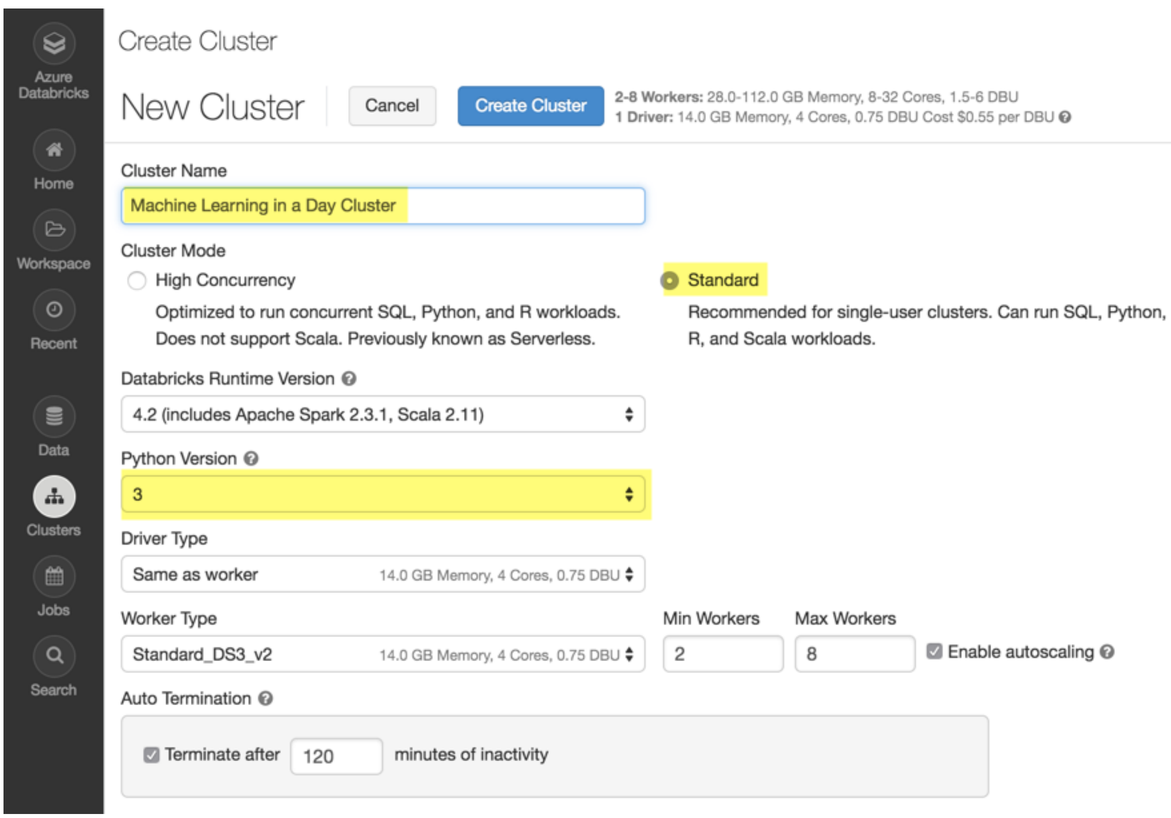
4. You will then be taken to the Databricks workspace and signed in automatically through your **Azure Active Directory** credentials as seen in the following screenshot:



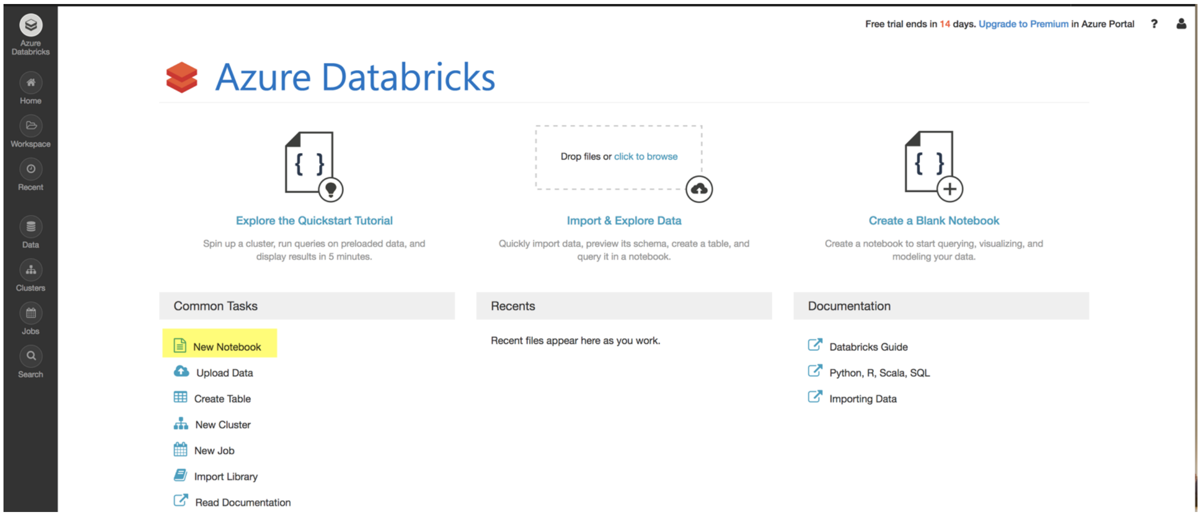
5. You will then land in the **Azure Databricks** home page. The first thing we will need to do is provision a **new cluster**, as seen in the following screenshot:



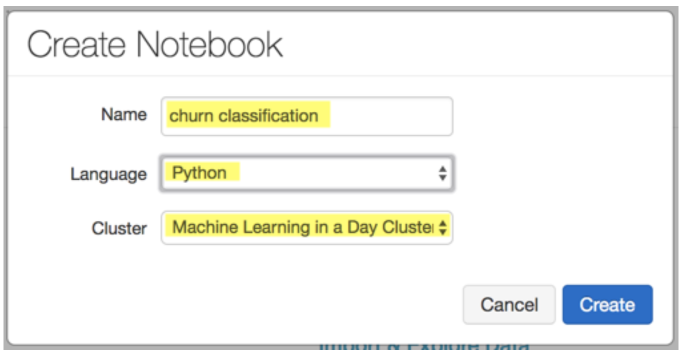
6. For our purposes for this workshop, we will provision a **Standard** mode cluster with **Python Version 3** support called **Machine learning in a Day Cluster** as seen in the following screenshot:



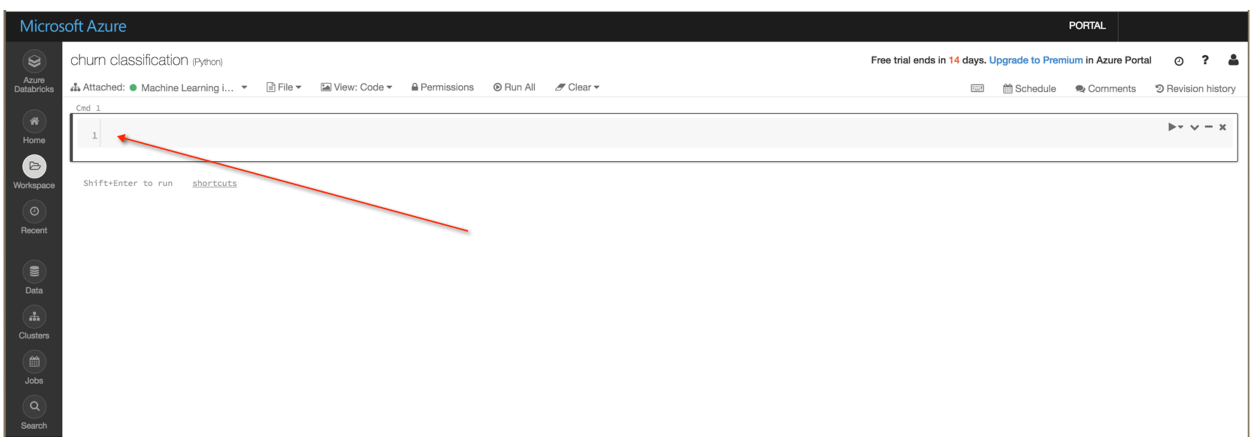
7. Once we select **Create Cluster**, it will begin provisioning and we can move onto creating our first notebook as seen in the following screenshot:



8. We name our notebook, **churn classification**, and assign it a default language of **Python** with the cluster we just created as seen in the following screenshot:



9. We are now in our notebook where we will be building our machine learning classification model in cells as seen in the following screenshot:



10. We are going to construct our URL to our blob file. This will requires using the SAS URL link that we coped from step 26 from the previous section. We will set the URL to a variable and read it into a Spark DataFrame using the following script in the cell:

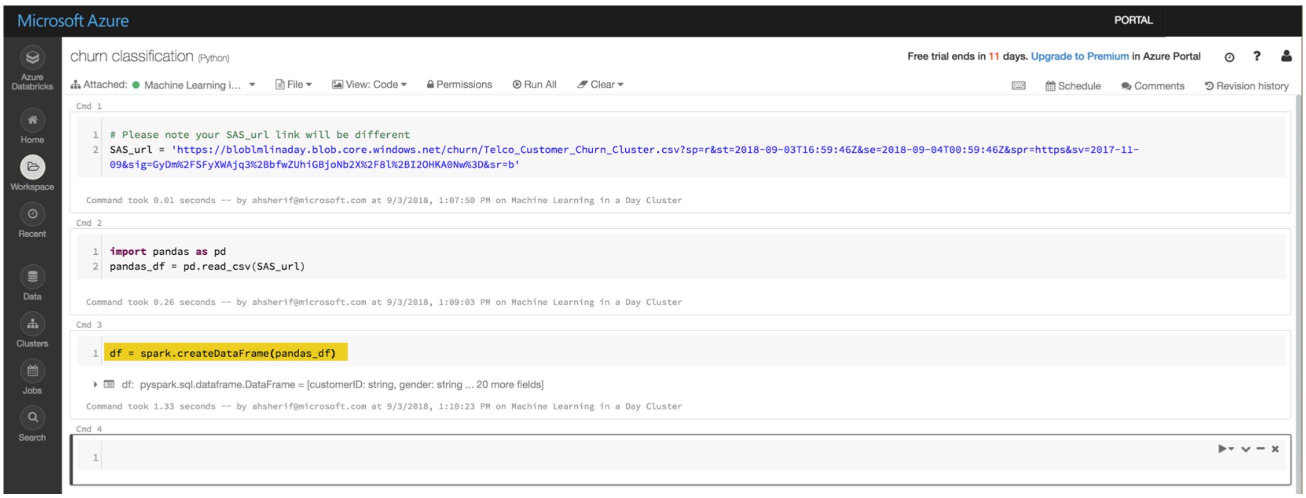
```
SAS_url =
'https://bloblmlinaday.blob.core.windows.net/churn/Telco_Customer_Churn_Cluster.csv
?sp=r&st=2018-09-03T16:59:46Z&se=2018-09-04T00:59:46Z&spr=https&sv=2017-11-
09&sig=GyDm%2FSFyXWAjq3%2BbfwZUhiGBjoNb2X%2F81%2BI2OHKA0Nw%3D&sr=b'
```

Please note that your SAS_url will be different than the one in this documentation

11. Next we will read our dataset into a Python DataFrame known as pandas as seen in the following screenshot:

```
import pandas as pd
pandas_df = pd.read_csv(SAS_url)

df = spark.createDataFrame(pandas_df)
```

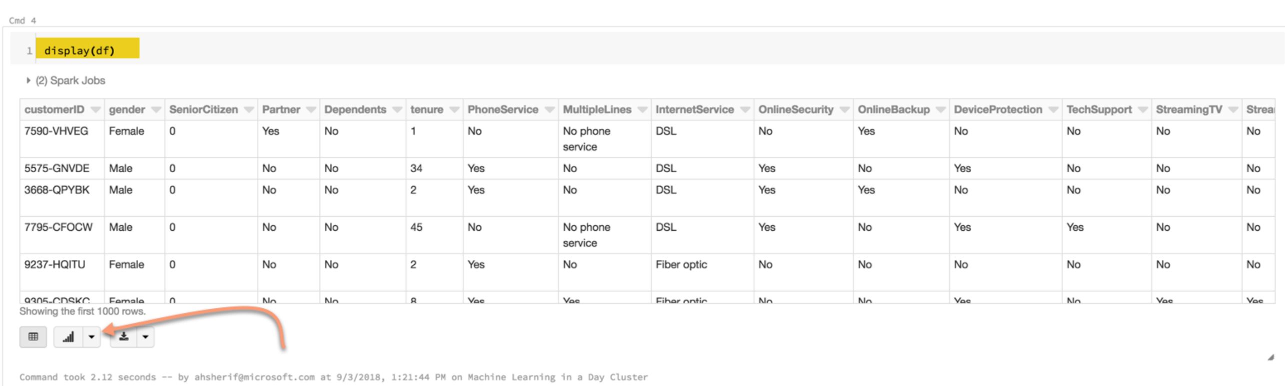
12. The concept of a Dataframe comes from the world of statistical software used in empirical research; it generally refers to "tabular" data: a data structure representing cases (rows), each of which consists of a number of observations or measurements (columns). Think of a spreadsheet in a CSV or Excel format or a table in a SQL database.
13. The script appears as the following in the notebook:



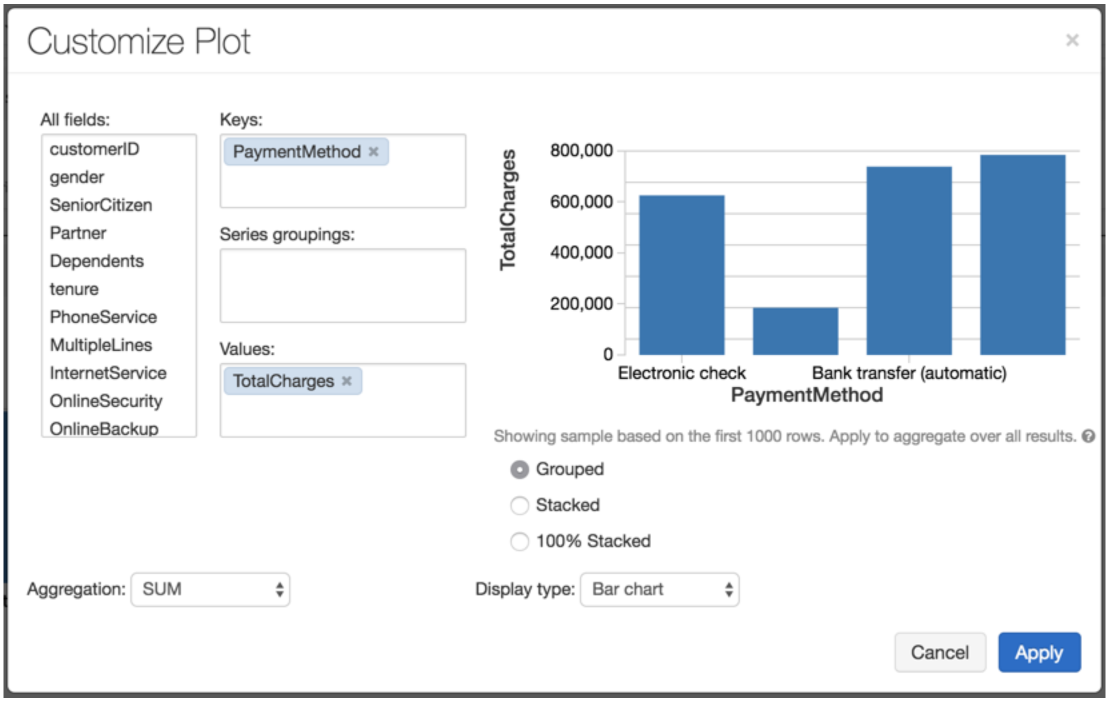
1. The easiest way to create a visualization in Databricks is to call the following script:

```
display(<dataframe-name>)
```

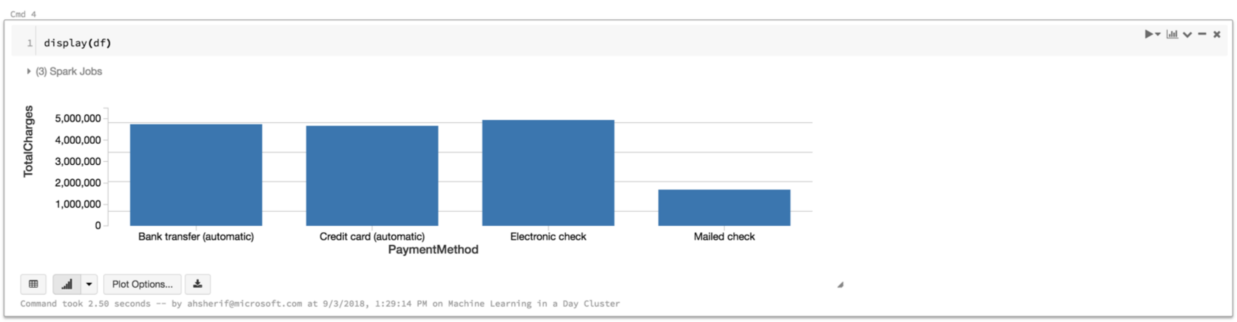
2. Our Spark DataFrame, **df**, is created and can be viewed by executing `display(df)` as seen in the following screenshot:



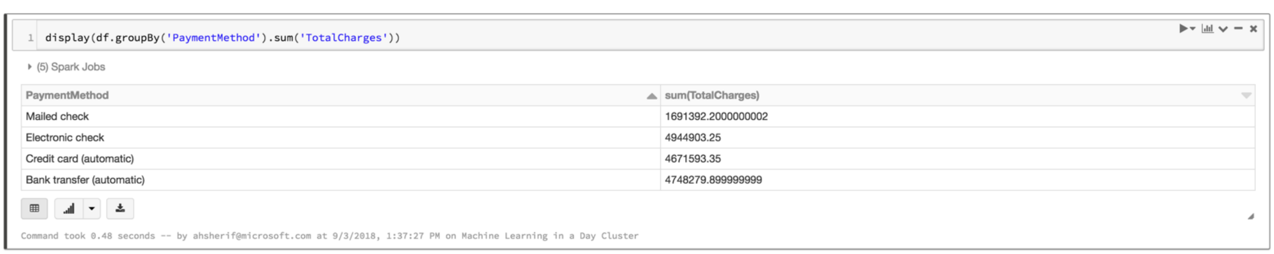
3. Databricks gives us the option to convert DataFrames into visualizations on the fly without having to code them using the bar chart icon underneath the DataFrame. For example, we can create a bar chart with **PaymentMethod** on the x-axis and **TotalCharges** on the y-axis by sum by clicking on the bar chart icon on the bottom of the DataFrame as seen in the following screenshot:



4. Once we apply the **keys**, **groupings**, and **values** we can see the output directly within the cell of the notebook as seen in the following screenshot:



5. Querying and manipulating DataFrames in Spark is not that hard to learn, especially if you have some existing skills in Python, SQL, or R. We can use Python syntax to do some data analysis and view **TotalCharges** by **PaymentMethod** using the following Python syntax script `display(df.groupBy('PaymentMethod').sum('TotalCharges'))` as seen in the following screenshot:



display(df.groupBy('PaymentMethod').sum('TotalCharges'))

(5) Spark Jobs

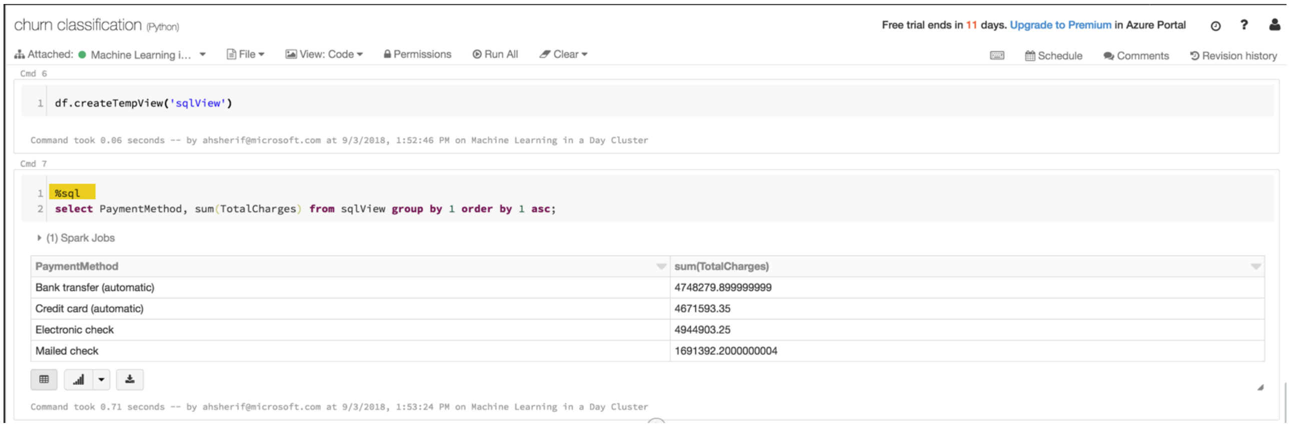
PaymentMethod	sum(TotalCharges)
Mailed check	1691392.2000000002
Electronic check	4944903.25
Credit card (automatic)	4671593.35
Bank transfer (automatic)	4748279.899999999

Command took 0.48 seconds -- by ahsherif@microsoft.com at 9/3/2018, 1:37:27 PM on Machine Learning in a Day Cluster

6. However, we can also do the same data analysis using SQL syntax. First, we create a view using the following script: `df.createTempView('sqlView')`.
7. Then we create a cell with SQL syntax and execute the following script:

```
%sql

select PaymentMethod, sum(TotalCharges) from sqlView group by 1 order by 1 asc;
```


8. The output of the script can be seen in the following screenshot:

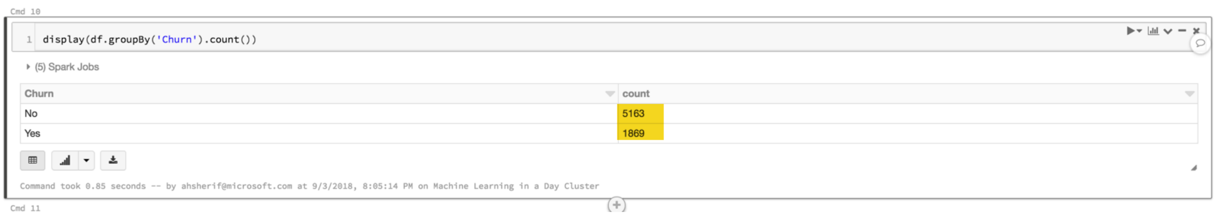


9. The results should match exactly what we had in the Python Script output. At this point we are ready to move on from just doing some data analysis with Spark on Databricks.

Section 6: Implement Featuring Engineering Techniques to Enhance data for Machine Learning

1. Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. It is often stated that feature engineering takes roughly 80% of the time from a data scientists day job.
2. The classification goal is to predict whether the customer will leave or churn (**Yes**/**No**).
3. We can take a look at our distribution of **Churn** by frequency to see what our distribution of churned customers we have in our sample set by executing the following script as seen in the following screenshot:

```
display(df.groupBy('Churn').count())
```



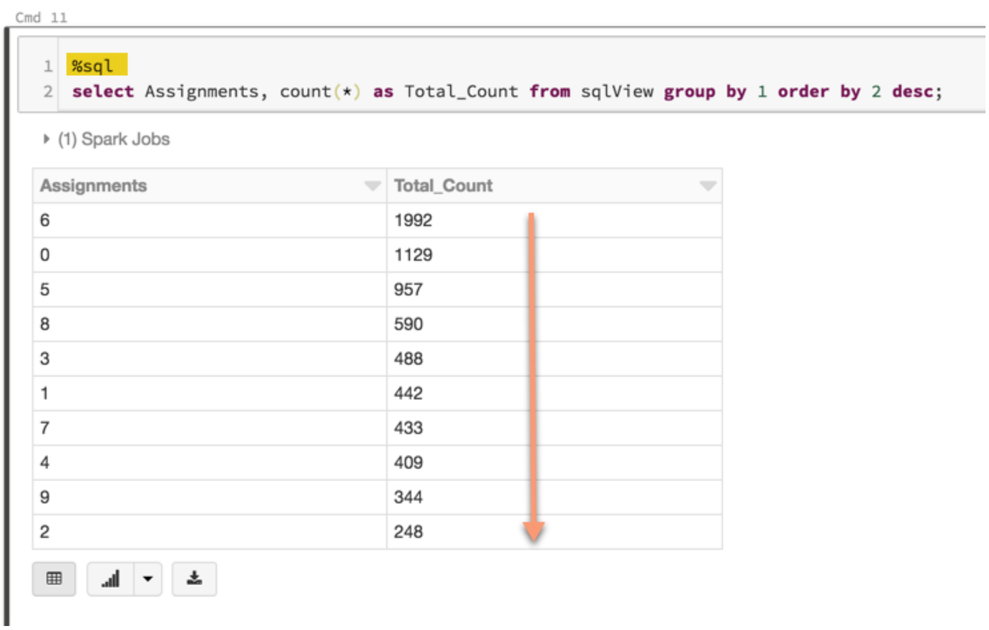
```
display(df.groupBy('Churn').count())
```

(5) Spark Jobs

Churn	count
No	5163
Yes	1869

4. So, about 27% or **1,869** of the customers that we have were churned versus 73% or **5,163** that remained with the company within the last year. Finally, we can take a look at the clusters that we created over in AML studio and see once again how they were distributed amongst the 10 clusters (0-9) that we created for them using the following script, , as seen in the following screenshot:

```
%sql
select Assignments, count(*) as Total_Count from sqlView group by 1 order by 2 desc;
```

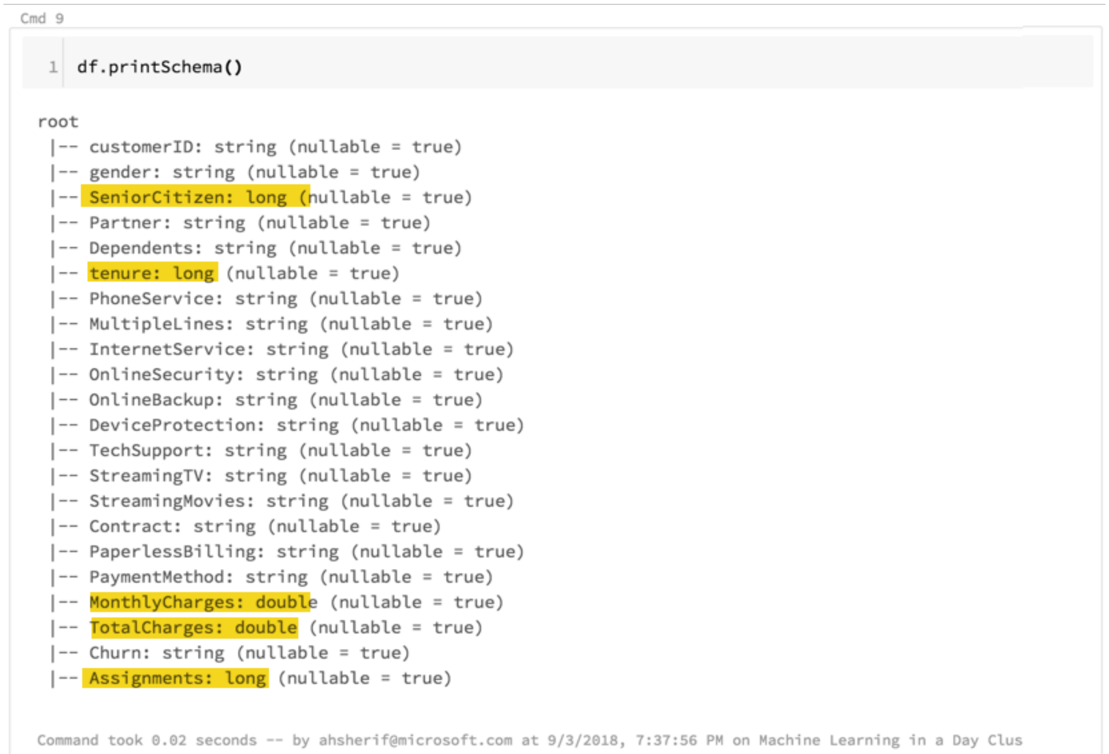


```
%sql
select Assignments, count(*) as Total_Count from sqlView group by 1 order by 2 desc;
```

(1) Spark Jobs

Assignments	Total_Count
6	1992
0	1129
5	957
8	590
3	488
1	442
7	433
4	409
9	344
2	248

5. We have the ability to see the schema of the data within the DataFrame by executing `df.printSchema()` as seen in the following screenshot:



```
Cmd 9

df.printSchema()

root
 |-- customerID: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: long (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- tenure: long (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- MonthlyCharges: double (nullable = true)
 |-- TotalCharges: double (nullable = true)
 |-- Churn: string (nullable = true)
 |-- Assignments: long (nullable = true)

Command took 0.02 seconds -- by ahsherif@microsoft.com at 9/3/2018, 7:37:56 PM on Machine Learning in a Day Clus
```

6. While most fields are string, we do have some that are **long** integer and some that are **double**. Other than **customerID**, all of the variables will be our independent variables (what we will use to predict) and **Churn** will be our dependent variable (what we are predicting).
7. For the purposes of machine learning, these string or categorical variables will need to be encoded to numeric variables. First we will start with the **Churn** column and assign a value of 0 for **No** and 1 for **Yes** using the following script as seen in the following screenshot:

```
from pyspark.sql import functions as F
df = df.withColumn('Churn', F.when(df['Churn']=='Yes', 1).otherwise(0))
```

```
from pyspark.sql import functions as F
df = df.withColumn('Churn', F.when(df['Churn']=='Yes', 1).otherwise(0))

df: pyspark.sql.dataframe.DataFrame
    customerID: string
    gender: string
    SeniorCitizen: long
    Partner: string
    Dependents: string
    tenure: long
    PhoneService: string
    MultipleLines: string
    InternetService: string
    OnlineSecurity: string
    OnlineBackup: string
    DeviceProtection: string
    TechSupport: string
    StreamingTV: string
    StreamingMovies: string
    Contract: string
    PaperlessBilling: string
    PaymentMethod: string
    MonthlyCharges: double
    TotalCharges: double
    Churn: integer
    Assignments: long

Command took 0.13 seconds -- by ahsherif@microsoft.com at 9/3/2018, 8:34:14 PM on Machine Learning in a Day Cluster
```

8. `pyspark.sql.functions` help apply SQL-type functionality to DataFrames for manipulation.
9. The **Churn** column is now converted to an **integer** data type from a **string** data type. We can identify all of our string or categorical columns by executing the following script and seeing the output in the following screenshot:

```
categorical_variables = [i[0] for i in df.dtypes if i[1] == 'string']
```

```
Cmd 13
categorical_variables = [i[0] for i in df.dtypes if i[1] == 'string']

Command took 0.01 seconds -- by ahsherif@microsoft.com at 9/3/2018, 8:51:36 PM on Machine Learning in a Day Cluster
Cmd 14
categorical_variables

Out[35]:
['customerID',
 'gender',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod']
Command took 0.01 seconds -- by ahsherif@microsoft.com at 9/3/2018, 8:51:44 PM on Machine Learning in a Day Cluster
```

10. We have 16 categorical variables, but only 15 of them are useful from a machine learning perspective as **customerID** does not provide any intrinsic value. Therefore, we can remove it from our list of columns by executing the following script as seen in the following screenshot:

```
categorical_variables = categorical_variables[1:]
categorical_variables
```

```
categorical_variables = categorical_variables[1:]
categorical_variables

Out[37]:
['gender',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod']
Command took 0.01 seconds -- by ahsherif@microsoft.com at 9/3/2018, 8:55:35 PM on Machine Learning in a Day Cluster
```

11. **StringIndexer** encodes a string column of labels to a column of label indices. The indices are in [0, numLabels), ordered by label frequencies, so the most frequent label gets index 0. The unseen labels will be put at index numLabels if user chooses to keep them. If the input column is numeric, we cast it to string and index the string values.

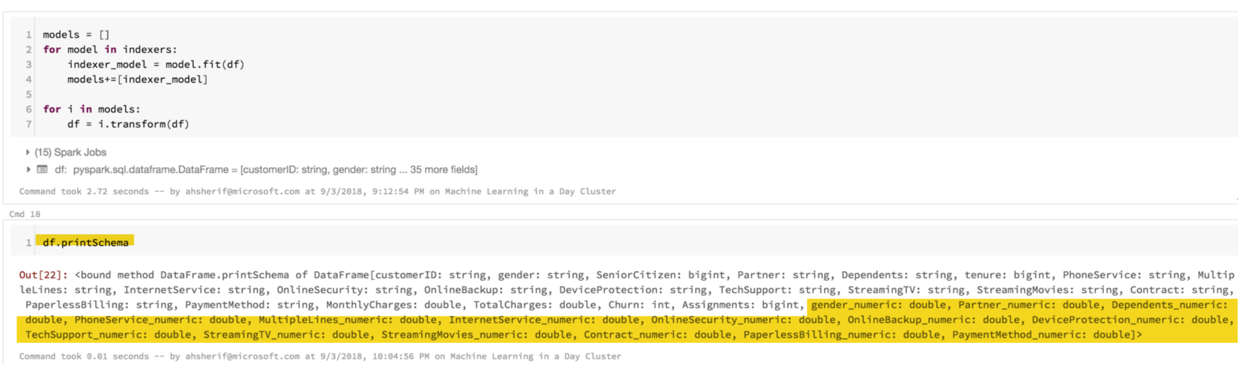
12. We import **StringIndexer** to transform each categorical column into a numeric column with a ***_numeric*** suffix using the following script (Python Script is whitespace sensitive, so keep the formatting on the tabs and indents, otherwise you will receive an error):

```
from pyspark.ml.feature import StringIndexer
indexers = []
for categoricalCol in categorical_variables:
    stringIndexer = StringIndexer(inputCol=categoricalCol, 
outputCol=categoricalCol+"_numeric")
    indexers += [stringIndexer]
```

13. Next, we will apply the transformation on the DataFrame using the following script:

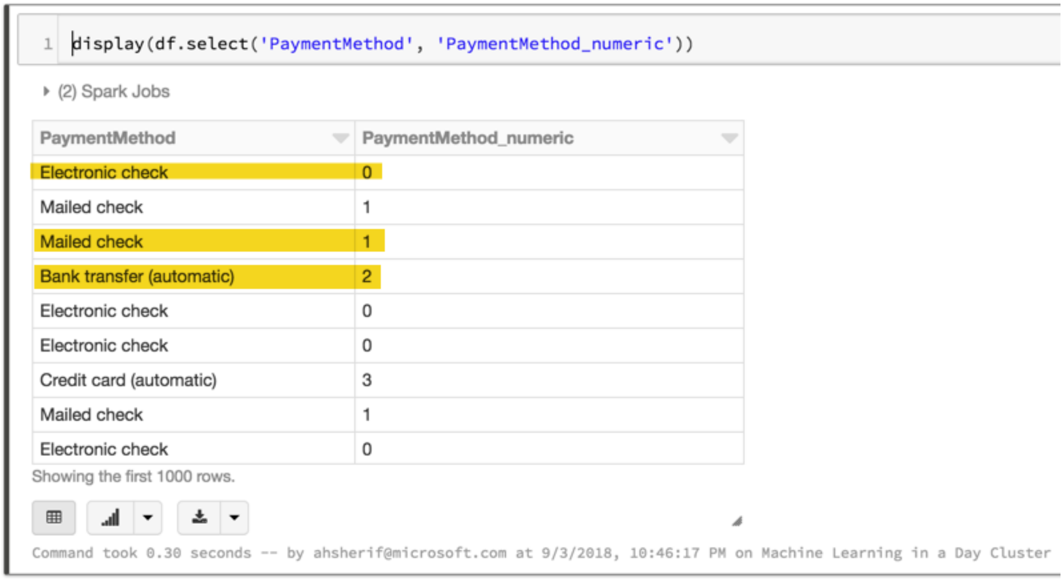
```
models = []
for model in indexers:
    indexer_model = model.fit(df)
    models+=[indexer_model]

for i in models:
    df = i.transform(df)
```

14. The output of the DataFrame should now reveal the newly added columns when executing the `df.printSchema` script as seen in the following screenshot:



15. Every column ending with **_numeric** has a **double** precision data type. We can do a side-by-side comparison of one of the columns, `PaymentMethod`, to see how the **_numeric** column compares to the string column by executing the following script and seeing the output in the following screenshot:

```
display(df.select('PaymentMethod', 'PaymentMethod_numeric'))
```



```
1 display(df.select('PaymentMethod', 'PaymentMethod_numeric'))
```

(2) Spark Jobs

PaymentMethod	PaymentMethod_numeric
Electronic check	0
Mailed check	1
Mailed check	1
Bank transfer (automatic)	2
Electronic check	0
Electronic check	0
Credit card (automatic)	3
Mailed check	1
Electronic check	0

Showing the first 1000 rows.

Command took 0.30 seconds -- by ahsherif@microsoft.com at 9/3/2018, 10:46:17 PM on Machine Learning in a Day Cluster

16. We have two columns, **MonthlyCharges** and **TotalCharges**, that need to be normalized from their original values. They are considered continuous variables and not categorical variables.
17. Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly. For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modeling.
18. Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.
19. We can make these transformations using the following script as seen in the output in the following screenshot:

```
from pyspark.sql.functions import min, max

minMonthly = df.agg(min(df.MonthlyCharges)).head()[0]
maxMonthly = df.agg(max(df.MonthlyCharges)).head()[0]

minTotal = df.agg(min(df.TotalCharges)).head()[0]
maxTotal = df.agg(max(df.TotalCharges)).head()[0]


df = df.withColumn('MonthlyCharges_Normalized',
                (df.MonthlyCharges - minMonthly)/(maxMonthly-minMonthly))
df = df.withColumn('TotalCharges_Normalized',
                (df.TotalCharges - minTotal)/(maxTotal-minTotal))
```

```
from pyspark.sql.functions import min, max

minMonthly = df.agg(min(df.MonthlyCharges)).head()[0]
maxMonthly = df.agg(max(df.MonthlyCharges)).head()[0]

minTotal = df.agg(min(df.TotalCharges)).head()[0]
maxTotal = df.agg(max(df.TotalCharges)).head()[0]


df = df.withColumn('MonthlyCharges_Normalized', (df.MonthlyCharges - minMonthly)/(maxMonthly-minMonthly))
df = df.withColumn('TotalCharges_Normalized', (df.TotalCharges - minTotal)/(maxTotal-minTotal))
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [customerID: string, gender: string ... 22 more fields]

Command took 0.83 seconds -- by ahsherif@microsoft.com at 9/4/2018, 10:07:01 AM on Machine Learning in a Day Cluster

20. We can do a side-by-side comparison of the original **MonthlyCharges** column with the newly added **MonthlyCharges_Normalized** column by executing the following script and seeing the output in the following screenshot:

```
display(df.select('MonthlyCharges', 'MonthlyCharges_Normalized'))
```

(2) Spark Jobs

MonthlyCharges	MonthlyCharges_Normalized
29.85	0.11542288557213931
56.95	0.3850746268656717
53.85	0.35422885572139307
42.3	0.23930348258706466
70.7	0.5218905472636817
99.65	0.809950248756219
89.1	0.7049751243781094
29.75	0.11442786069651742

Showing the first 1000 rows.

Command took 0.21 seconds -- by ahsherif@microsoft.com at 9/4/2018, 10:07:04 AM on Machine Learning in a Day Cluster

21. We have now completed our feature engineering process and we are ready to assign the columns that we will use as the final features for making our machine learning model work as efficiently as possible. The final **features** will be selected using the following script:

```
features = [
 'Assignments',

 'gender_numeric',
 'Partner_numeric',
 'Dependents_numeric',
 'PhoneService_numeric',
 'MultipleLines_numeric',
 'InternetService_numeric',
 'OnlineSecurity_numeric',
 'OnlineBackup_numeric',
 'DeviceProtection_numeric',
 'TechSupport_numeric',
 'StreamingTV_numeric',
 'StreamingMovies_numeric',
 'Contract_numeric',
 'PaperlessBilling_numeric',
```

```
    'PaymentMethod_numeric',

    'MonthlyCharges_Normalized',
    'TotalCharges_Normalized',

    'SeniorCitizen',
    'tenure'
   ]
```

22. The output of the new list of columns can be seen in the following screenshot:

```
1  features

Out[19]:
['Assignments',
 'gender_numeric',
 'Partner_numeric',
 'Dependents_numeric',
 'PhoneService_numeric',
 'MultipleLines_numeric',
 'InternetService_numeric',
 'OnlineSecurity_numeric',
 'OnlineBackup_numeric',
 'DeviceProtection_numeric',
 'TechSupport_numeric',
 'StreamingTV_numeric',
 'StreamingMovies_numeric',
 'Contract_numeric',
 'PaperlessBilling_numeric',
 'PaymentMethod_numeric',
 'MonthlyCharges_Normalized',
 'TotalCharges_Normalized',
 'SeniorCitizen',
 'tenure']
```

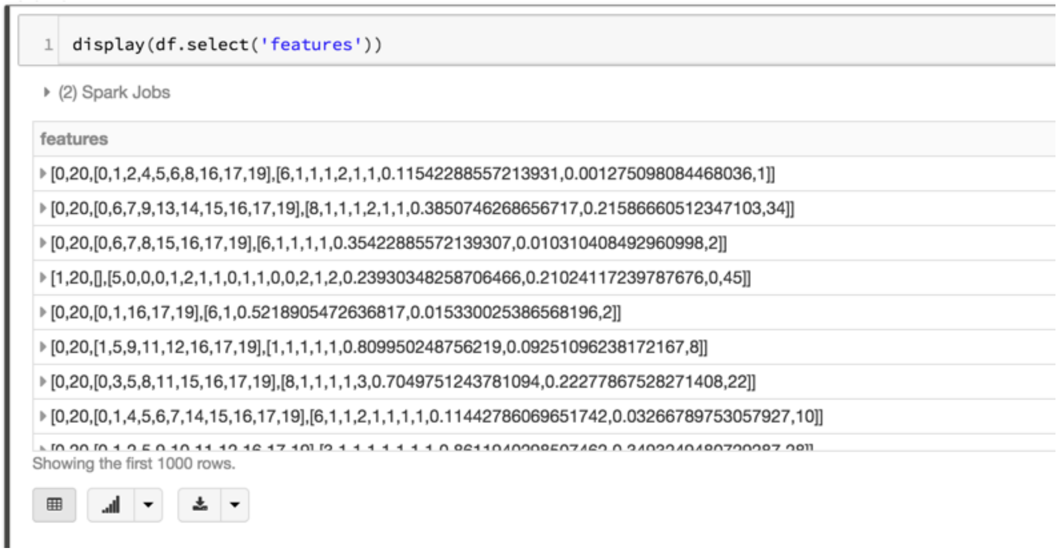
23. We will apply **VectorAssembler** to all of our features. VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression. We can specify which columns will be vectorized by applying the following script:

```
from pyspark.ml.feature import VectorAssembler

feature_vectors = VectorAssembler(
        inputCols = features,
        outputCol = "features")
```

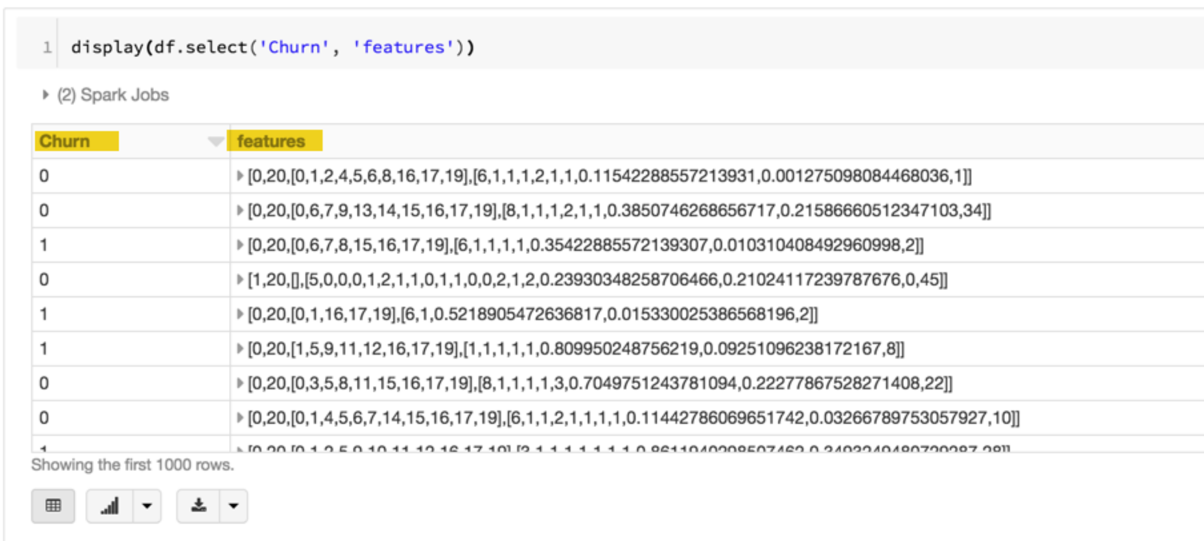
24. We can transform the DataFrame to include the newly created column, **features**, by executing the following script:

```
df = feature_vectors.transform(df)
```

25. We can the output of the field, **features**, by executing `display(df.select('features'))`, as seen in the following screenshot:



26. There are only two columns now that are of interest to us: **Churn** and **features**. **Churn** is our label column that we are trying to predict and **features** is our predictor column that is combined with all of our predictor variables in one vector as seen in the following screenshot:



27. One final modification we will make is convert the name **Churn** into **label** as it will better generalize the purpose of that field. We execute the following script, `df = df.withColumnRenamed('Churn', 'label')`, to rename the existing field to **label**. The output of the script can be seen by scrolling down to the location where **Churn** was and now seeing it replaced with **label** as seen in the following screenshot:

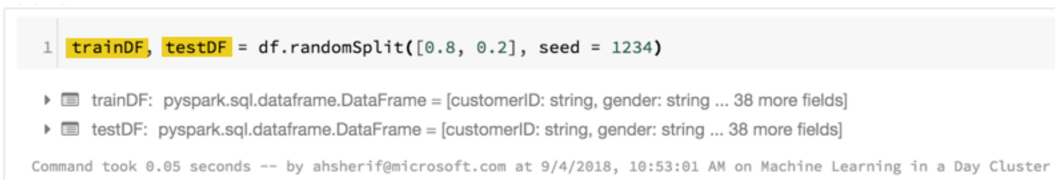
```
1  df = df.withColumnRenamed('Churn', 'label')
```

```
▾ df: pyspark.sql.dataframe.DataFrame
      MonthlyCharges: double
      TotalCharges: double
      label: integer
      Assignments: long
      MonthlyCharges_Normalized: double
      TotalCharges_Normalized: double
      gender_numeric: double
      Partner_numeric: double
      Dependents_numeric: double
      PhoneService_numeric: double
      MultipleLines_numeric: double
      InternetService_numeric: double
      OnlineSecurity_numeric: double
      OnlineBackup_numeric: double
      DeviceProtection_numeric: double
      TechSupport_numeric: double
      StreamingTV_numeric: double
      StreamingMovies_numeric: double
      Contract_numeric: double
      PaperlessBilling_numeric: double
      PaymentMethod_numeric: double
    ▾ features: udt
```

Section 7: Apply Classification Supervised Model with Logistic Regression on Azure Databricks

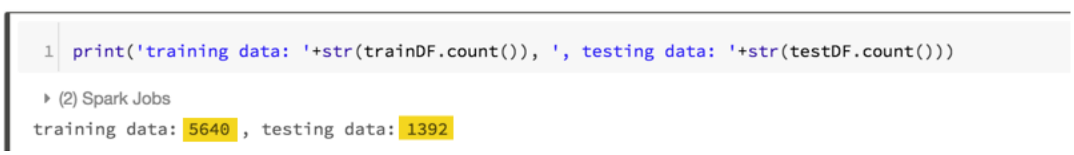
1. Logistic regression is a method for classifying data into discrete outcomes. For example, we might use logistic regression to classify a customer as **Y** or **N** for Churn.
2. In machine learning we usually split our data into two subsets: training data and testing data (sometimes we even split them into three: train, validate and test), and fit our model on the train data, in order to make predictions on the test data. This is used to help solve the problem of overfitting. Overfitting is the problem of fitting your model so well to the data that you have but not taking into consideration future data points. It lacks generality.
3. We split our data into a testing and training dataset using the following script with the output in the following screenshot:

```
trainDF, testDF = df.randomSplit([0.8, 0.2], seed = 1234)
```



4. The randomness is set to **1234** for reproducibility.
5. We can see the number of rows from our original DataFrame assigned to test versus train by executing the following script with the output seen in the following screenshot:

```
print('training data: '+str(trainDF.count()), ', testing data:
'+str(testDF.count()))
```



6. Next we want to build out our **logistic regression classification** model pipeline and **fit** it on our training DataFrame using the following script:

```
from pyspark.ml.classification import LogisticRegression
logreg = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10)
LogisticRegressionModel = logreg.fit(trainDF)
```

7. Next we test the model against our test DataFrame, **testDF**, using the following script to create a new DataFrame called **predictedDF**:

```
predictedDF = LogisticRegressionModel.transform(testDF)
```

8. The output of our predicted values along with the original label can be seen by executing the following script with the output in the following screenshot:

```
display(predictedDF.select('label', 'probability', 'prediction'))
```

```
display(predictedDF.select('label', 'probability', 'prediction'))
```

▸ (3) Spark Jobs

label	probability	prediction
0	▸ [1,2,[],[0.2453414700108091,0.7546585299891908]]	1
0	▸ [1,2,[],[0.8531850801244731,0.1468149198755269]]	0
0	▸ [1,2,[],[0.9156653394194328,0.0843346605805672]]	0
0	▸ [1,2,[],[0.5615028194673514,0.4384971805326486]]	0
1	▸ [1,2,[],[0.5183021325512303,0.4816978674487698]]	0
1	▸ [1,2,[],[0.6351426553451417,0.3648573446548583]]	0
1	▸ [1,2,[],[0.5640926911816028,0.4359073088183972]]	0
0	▸ [1,2,[],[0.873286076625449,0.12671392337455098]]	0

Showing the first 1000 rows.

9. In the field of machine learning, a *confusion matrix*, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.
10. We can create a confusion matrix to identify *False Positives, False Negatives, True Positives, and True Negatives* using the following script with the output seen in the following screenshot:

```
display(predictedDF.crosstab('label', 'prediction'))
```

```
display(predictedDF.crosstab('label', 'prediction'))
```

▸ (5) Spark Jobs

label_prediction	0.0	1.0
1	171	206
0	910	105

11. We made **910** + **206** = 1,116 correct predictions and **171+105** = 276 false predictions for an overall accuracy of ~ **80%**
12. We can calculate the accuracy of the model by executing the following script and view the output as seen in the following screenshot:

```
from sklearn import metrics

actual = predictedDF.select('label').toPandas()
predicted = predictedDF.select('prediction').toPandas()
print('accuracy score = {}'.format(metrics.accuracy_score(actual, predicted)*100))
```

```
from sklearn import metrics

actual = predictedDF.select('label').toPandas()
predicted = predictedDF.select('prediction').toPandas()
print('accuracy score = {}'.format(metrics.accuracy_score(actual, predicted)*100))
```

▸ (2) Spark Jobs

```
accuracy score = 80.17241379310344
```

13. In addition, the *ROC* (Receiver Operating Characteristic) Curve or Area Under the Curve is another way to measure the performance of the model. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm[1] and can be calculated as (1 − specificity).
14. We can calculate the ROC or Area Under the Curve by executing the following script and see the output as seen in the following screenshot:

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print('ROC score = {}'.format(round(evaluator.evaluate(predictedDF)*100,2)))
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print('ROC score = {}'.format(round(evaluator.evaluate(predictedDF)*100,2)))
```

▸ (4) Spark Jobs

```
ROC score = 85.32
```

15. We have a high ROC score of **85.32**%. The higher the score or closer it is to 100% the better.

Section 8: Export predicted DataFrame from Spark on Azure Databricks to Power BI for visualizations

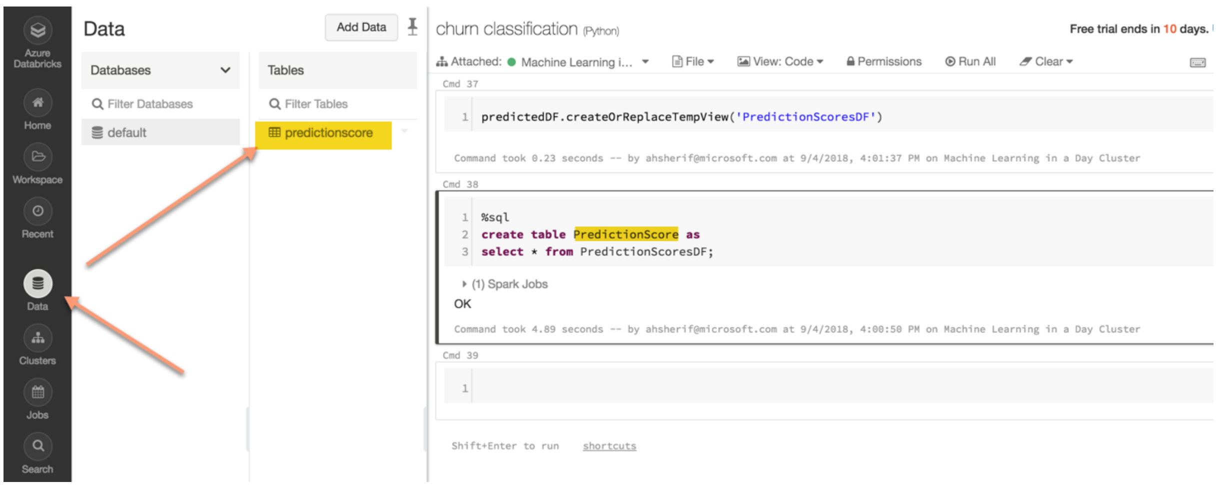
1. We are able to stream our DataFrames from Spark on Azure Databricks to other Business Intelligence tools such as Power BI, Tableau, and Alteryx.
2. First, we will create another view that called **PredictionScoresDF** using the following script:

```
predictedDF.createOrReplaceTempView('PredictionScoresDF')
```

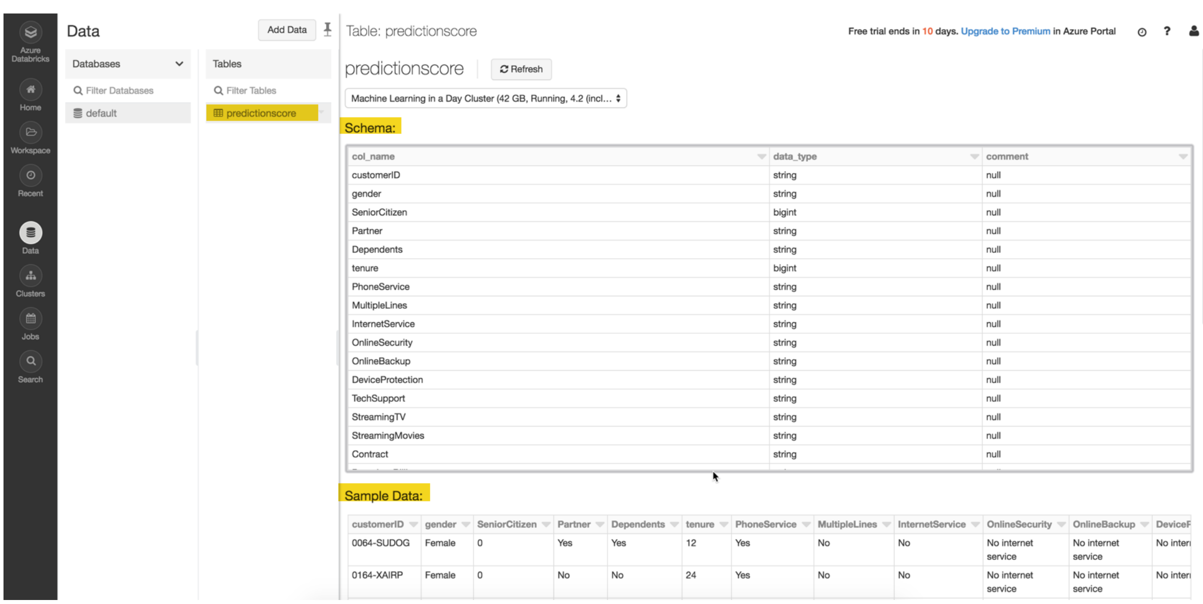
3. Next we will create a table, PredictionScore, accessible outside of Spark using the following script:

```
%sql
create table PredictionScore as
select * from PredictionScoresDF;
```

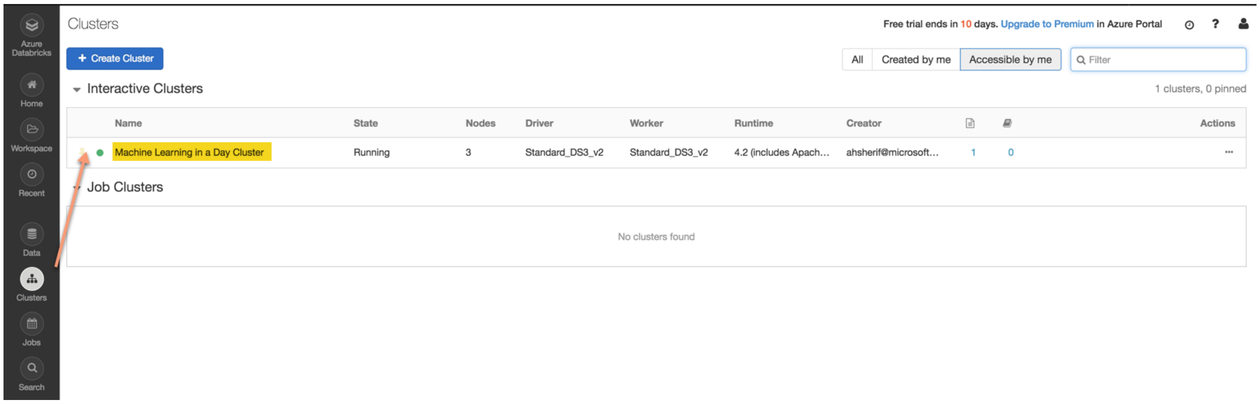
4. We can then see this table, **PredictionScore**, outside of the notebook and within the Data section within Databricks as seen in the following screenshot:



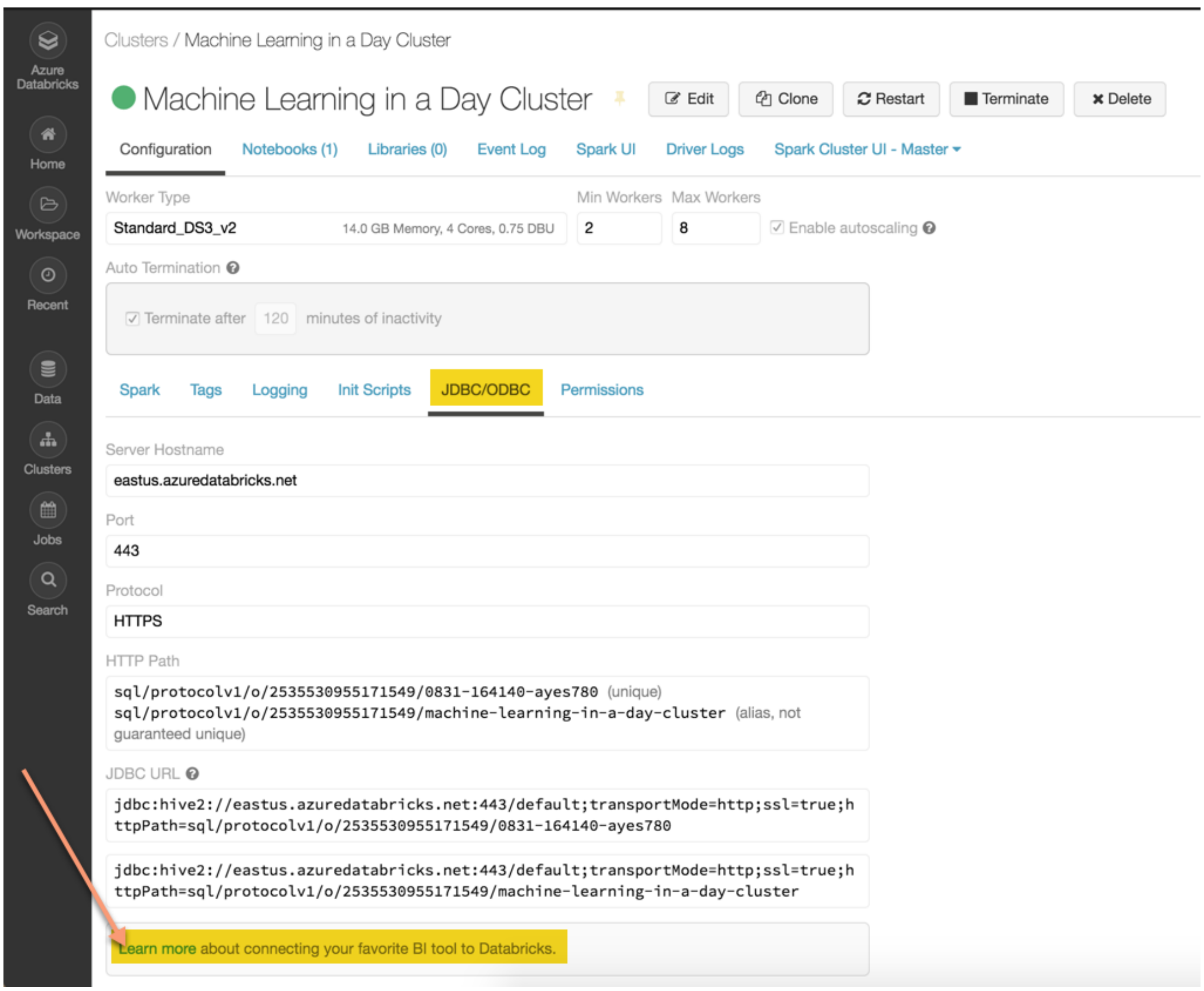
5. When clicking on the table, **predictionscore**, within Databricks, we can view both the **sample data** as well as the **schema** inside of the table as seen in the following screenshot:



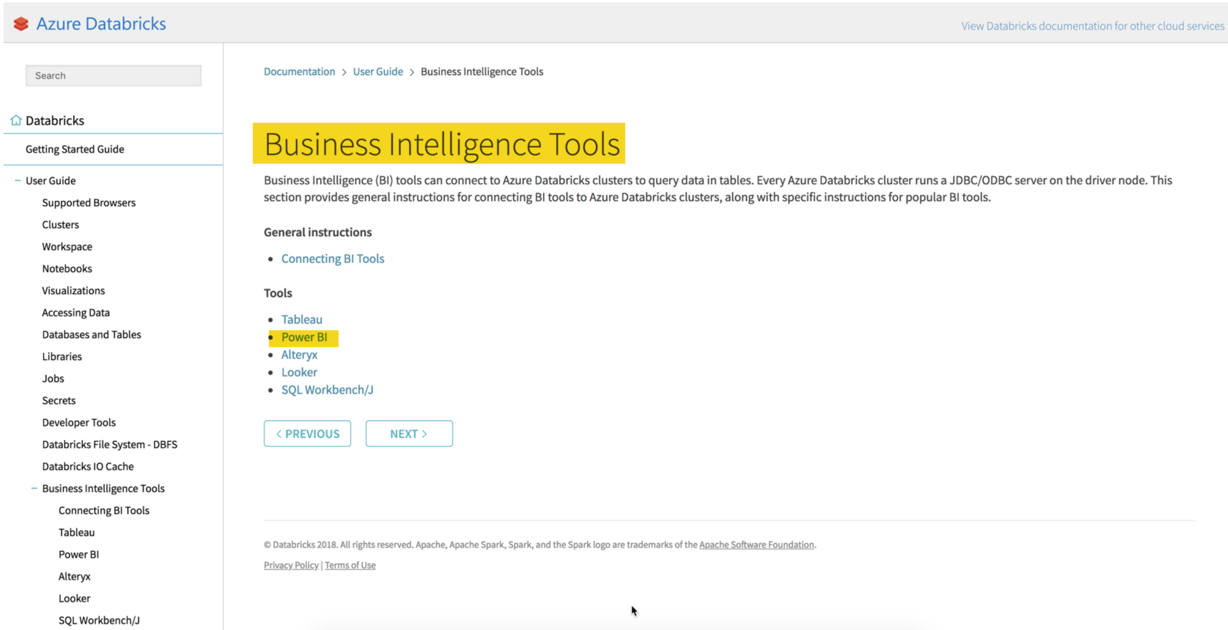
6. Next we will click on the **Clusters** icon on the left-hand side of the menu and select our current cluster, **Machine Learning in a Day Cluster**, as seen in the following screenshot:



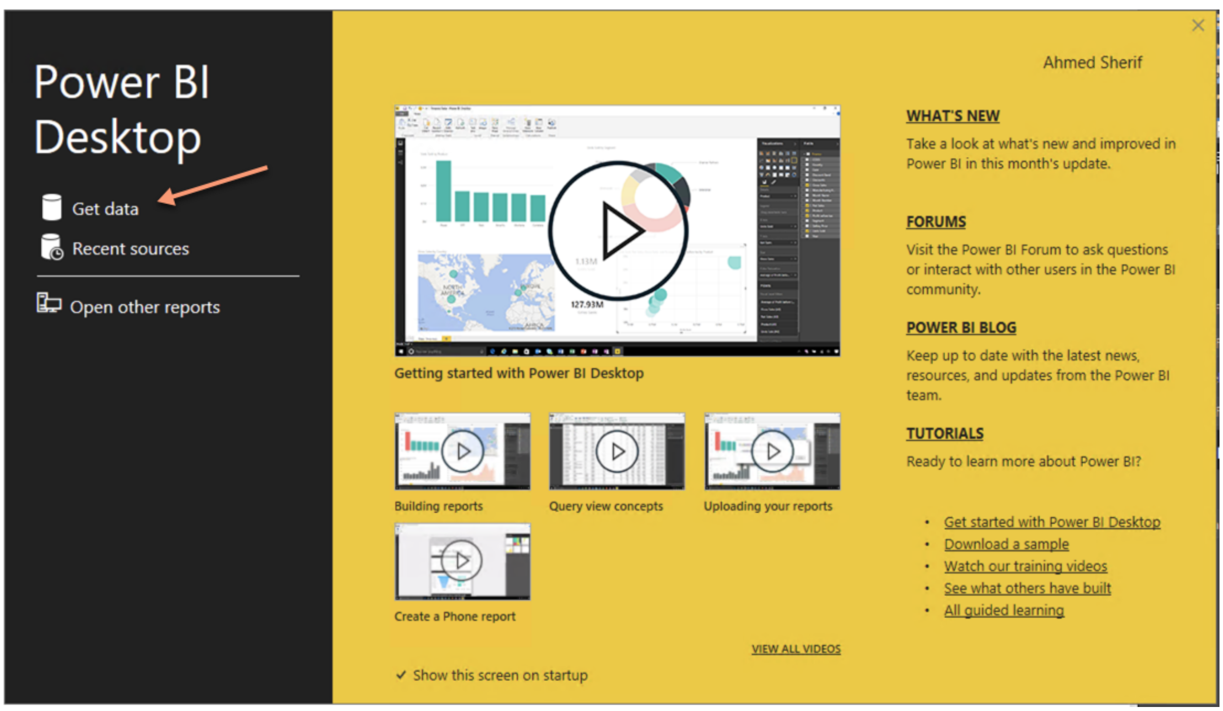
7. Once inside the cluster, click on the **JDBC/ODBC** tab and click on the link in the bottom **To Learn More About Connecting your Favorite BI Tools to Databricks**.



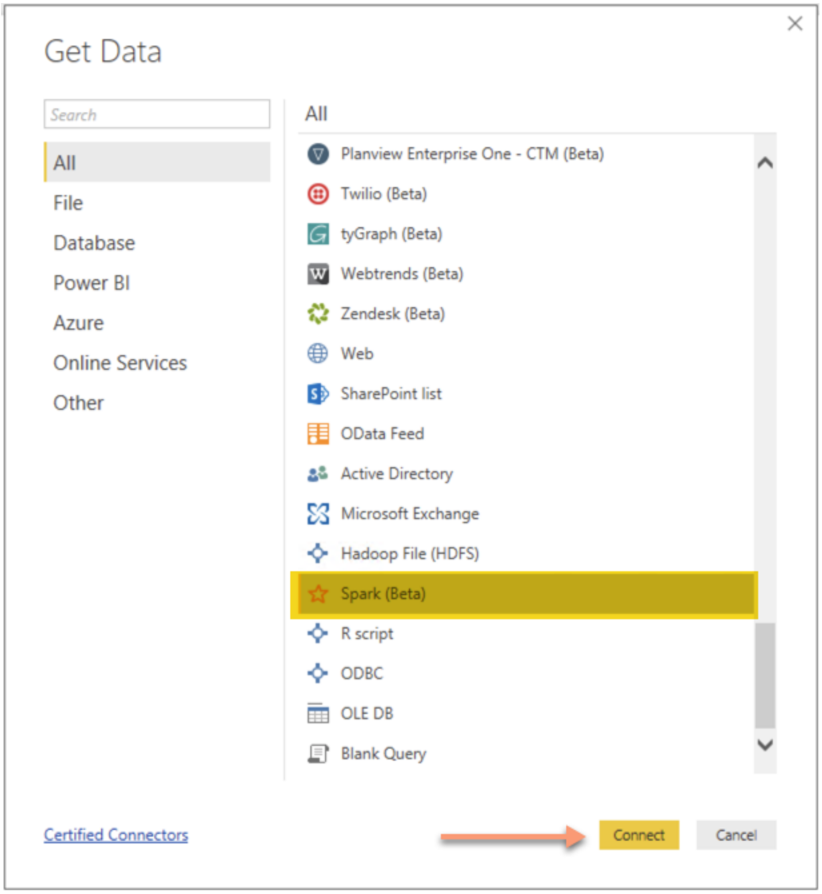
8. Select **Power BI** as the **Business Intelligence tool** of choice for connecting to Databricks as seen in the following screenshot:



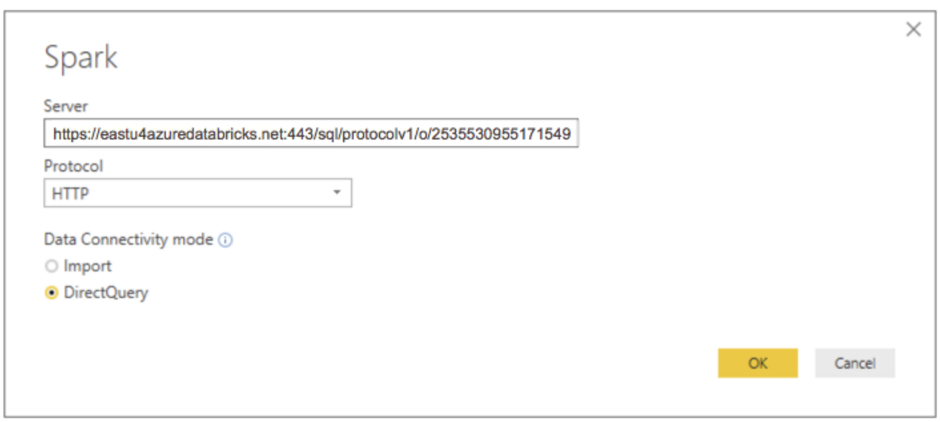
9. Databricks should have step-by-step directions for connecting Tables from Spark to Power BI as found in the following link: https://docs.azuredatabricks.net/user-guide/bi/power-bi.html
10. There should be a Power BI desktop installation already available in our Data Science Virtual machine that we can use. When we sign into Power BI, we will first select **Get Data** as seen in the following screenshot:



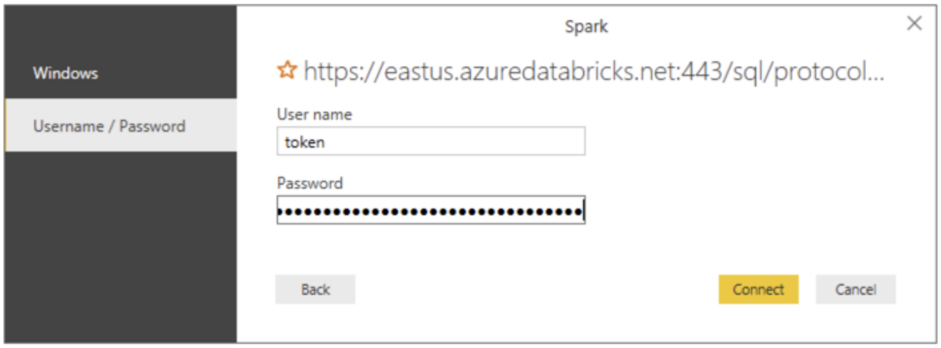
11. Select **Spark (Beta)** and **Connect** as seen in the following screenshot:



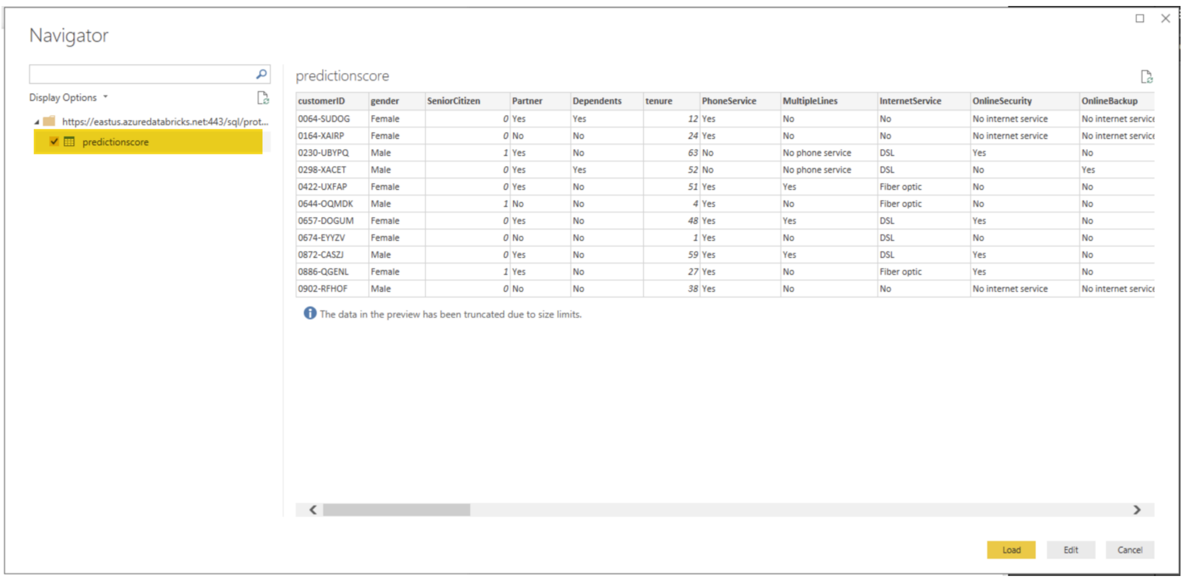
12. Enter the **Server** and **Protocol** Type as well as **DirectQuery** as the **data connectivity mode** as seen in the following screenshot:



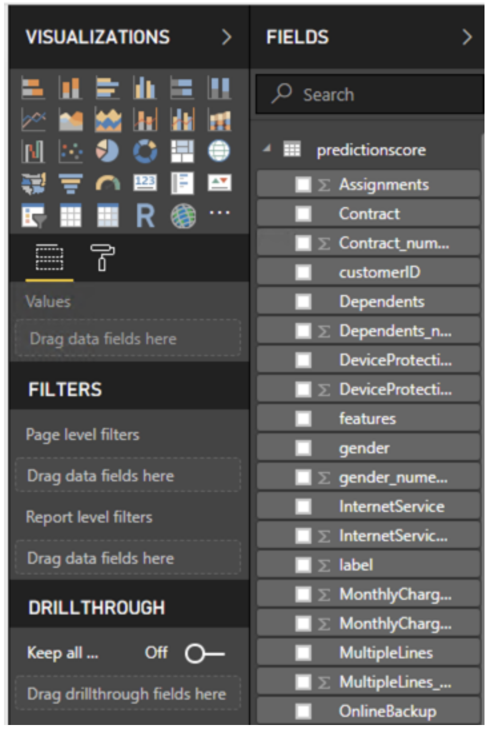
13. Next, Enter the User Name and Password credentials as seen in the following screenshot:



14. We can now view our data from the **predictionscore** table in Spark now viewed directly within Power BI as seen in the following screenshot:



15. Once we select **Load**, the dataset with all of the columns will be available for us to visualize inside of Power BI as seen in the following screenshot:



16. We can then create a simple visualization of a bar chart using **MonthlyCharges by Assignment** as seen in the following screenshot: