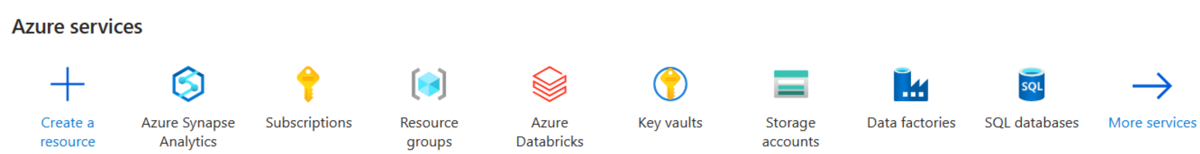
Integrating Machine Learning and Synapse

Created by	Ⓓ Devansh Gupta
Created time	@September 15, 2023 10:54 AM

STEP 1: Creation of Resource Group

Create a resource group by selecting the Resource Groups option in Azure Services. Name it according to convention.



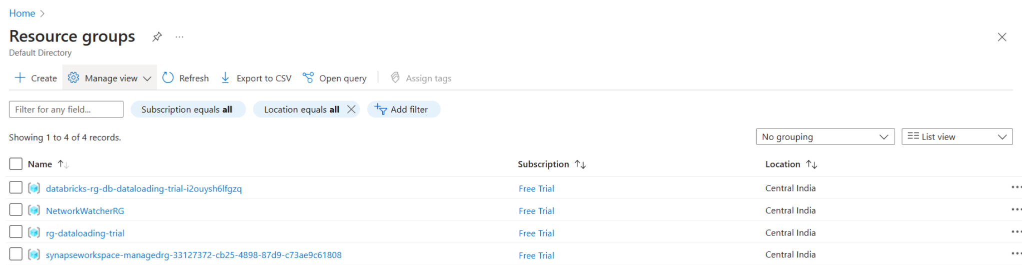
Then add all the below listed resources in the resource group -

1. Azure Synapse Analytics
2. Azure Blob Storage
3. Azure Machine Learning Studio

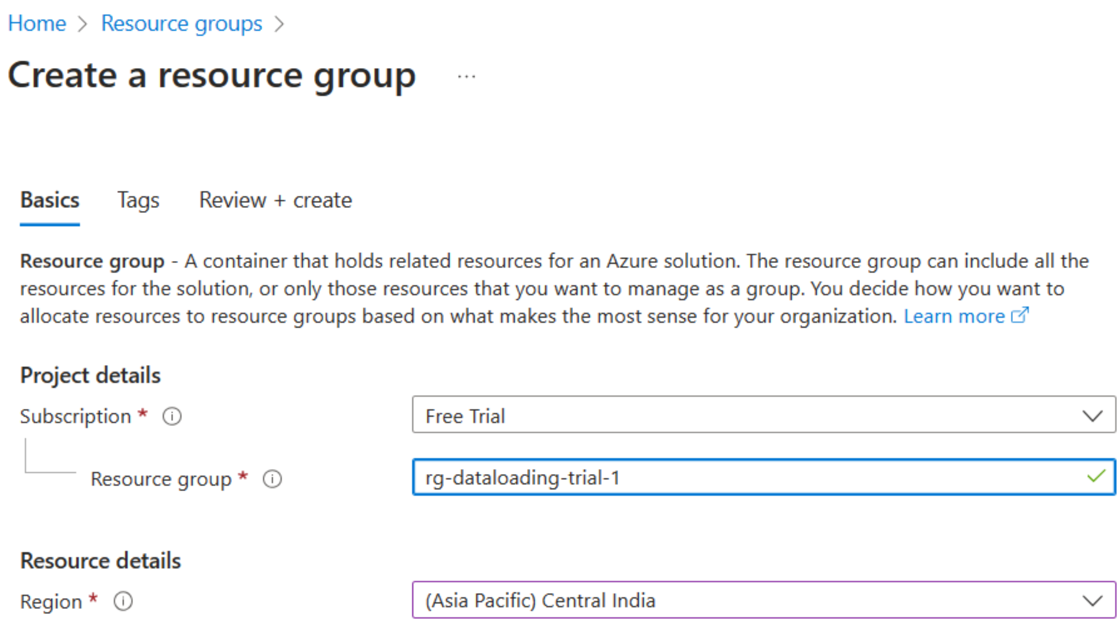
Many of these services are chargeable and thus, should be used carefully.

Ⅰ.Creating resource groups

1. Sign in to the Azure portal.
2. Select **Resource groups**.
3. Select **Create**



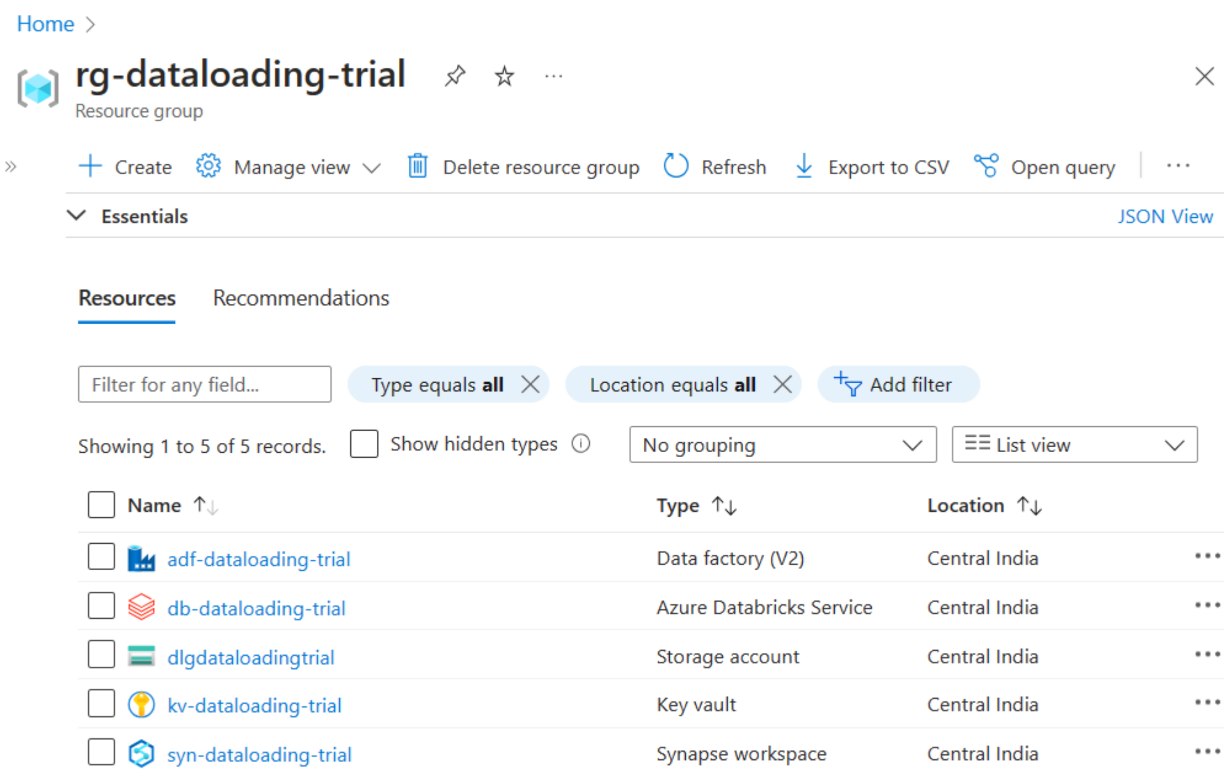
4. Enter the following values:
   - **Subscription**: Select your Azure subscription.
   - **Resource group**: Enter a new resource group name.
   - **Region**: Select an Azure location, such as **Central India**



5. Select **Review + Create**
6. Select **Create**. It takes a few seconds to create a resource group.

II .Adding Resouces to Resource Groups

Add all resources as shown below -



The settings that should be applied while creating each resource are plenty straightforward as should be applied correctly. Videos on YouTube can be referred to see the correct configuration of each resource.

STEP 2: Setting up the environment.

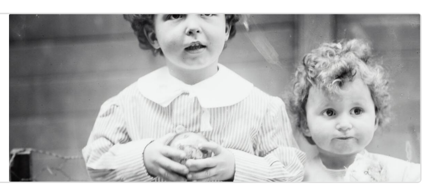
I . Finding the dataset on Kaggle

For this project, I have used the Titanic Dataset, the link for which is given below -

Titanic dataset

Gender submission and test file merged

k https://www.kaggle.com/datasets/brendan45774/test-file



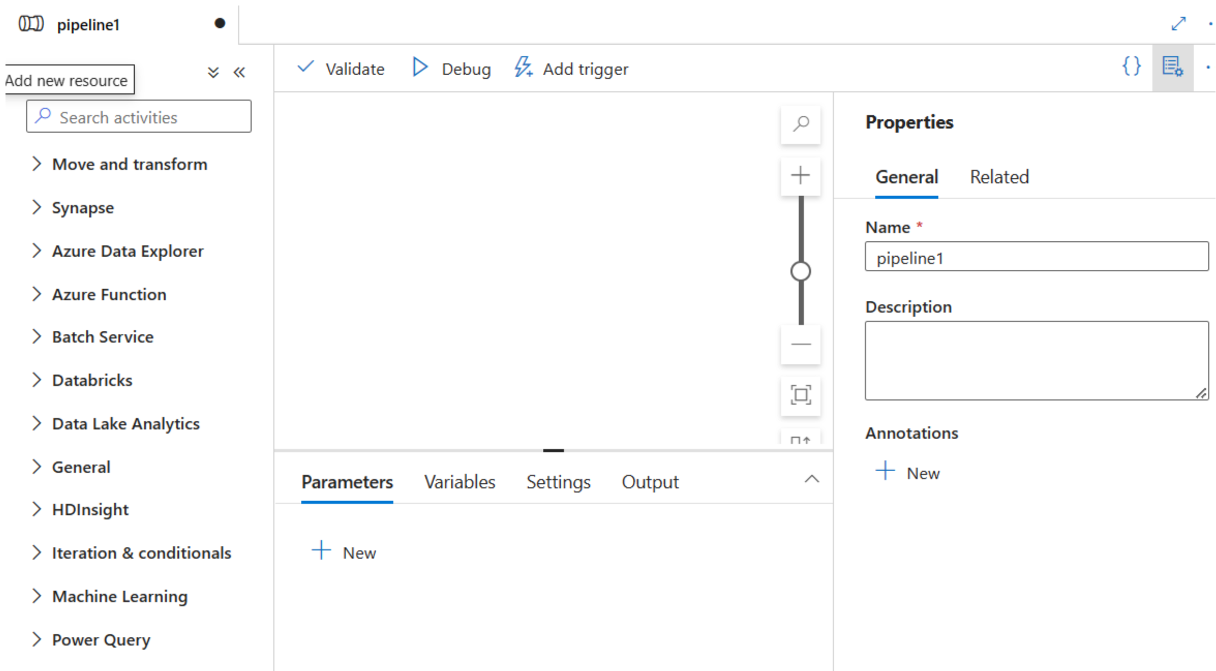
STEP 3: Data Ingestion.

1. Launch Azure Synapse Analytics Workspace.

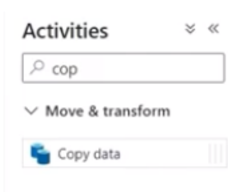
To create a new pipeline -

Author tab → Click on “+” icon → Select Pipeline → This creates a new pipeline.

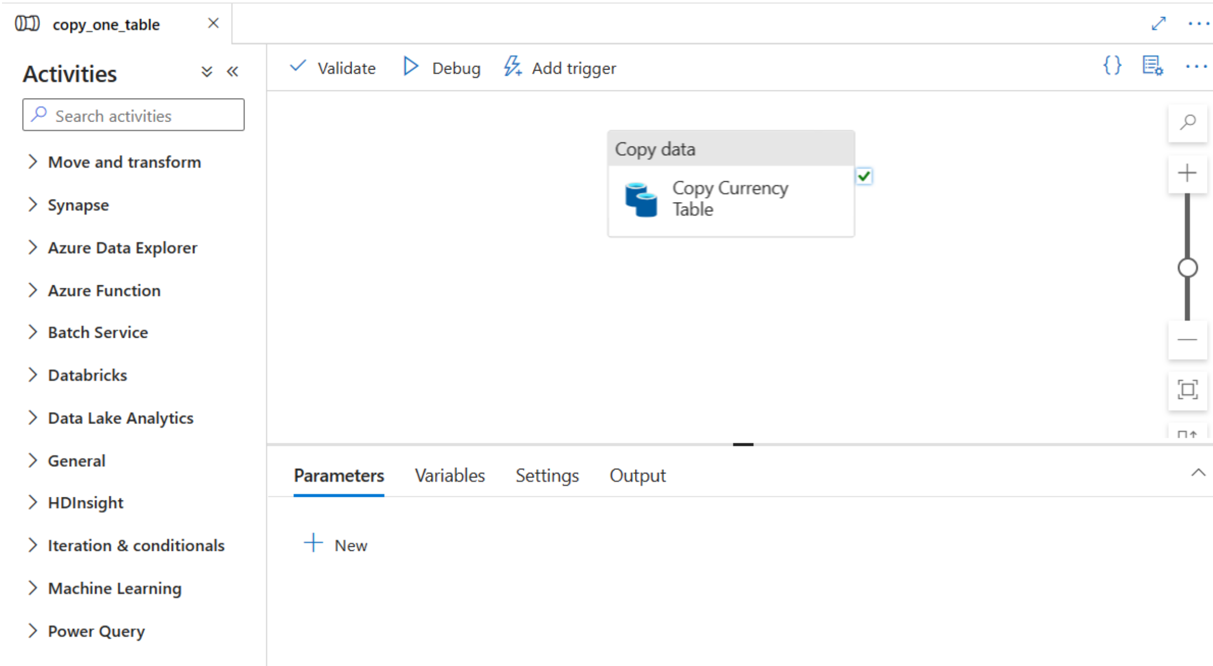
You can rename it if you want.



2. Now go to Activities and search for Copy data -

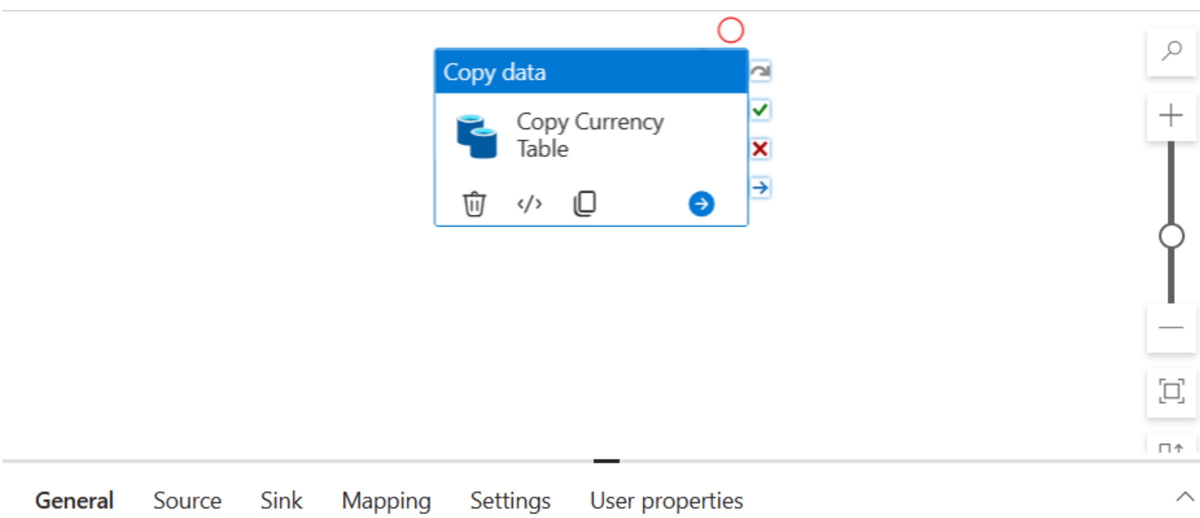


Now drag and drop this Activity in the workspace that you have been provided with.

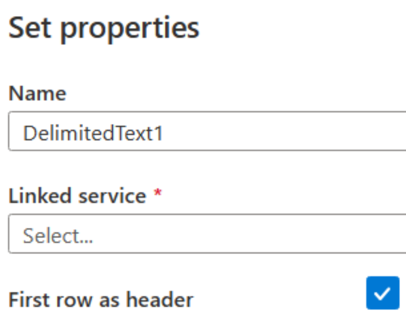


Also change the name of the Activity to make it more convenient to understand what it is doing.

3. To configure the activity, you have to configure the Source and Sink tabs respectively.



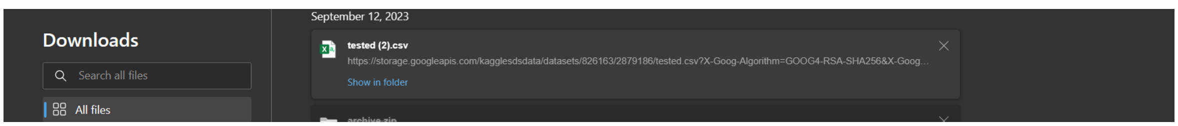
For Source dataset, click on +New → Then search for **HTTP** → Select the format as DelimitedText (**as our dataset is a csv file**) And name it as per your convenience. But the option below it - Linked Services have to be configured as done below -



Configuring Linked Service

1. Linked Service can be named as per convenience. This ensures connectivity of cloud and the on-premises database.
2. Connect via integration runtime - AutoResolveIntegrationRuntime (as we are connecting out synapse workspace directly to a online resource that is Kaggle)
3. For the Base URL - Copy and Paste the link of the dataset (**make sure you don't copy the link for kaggle**)

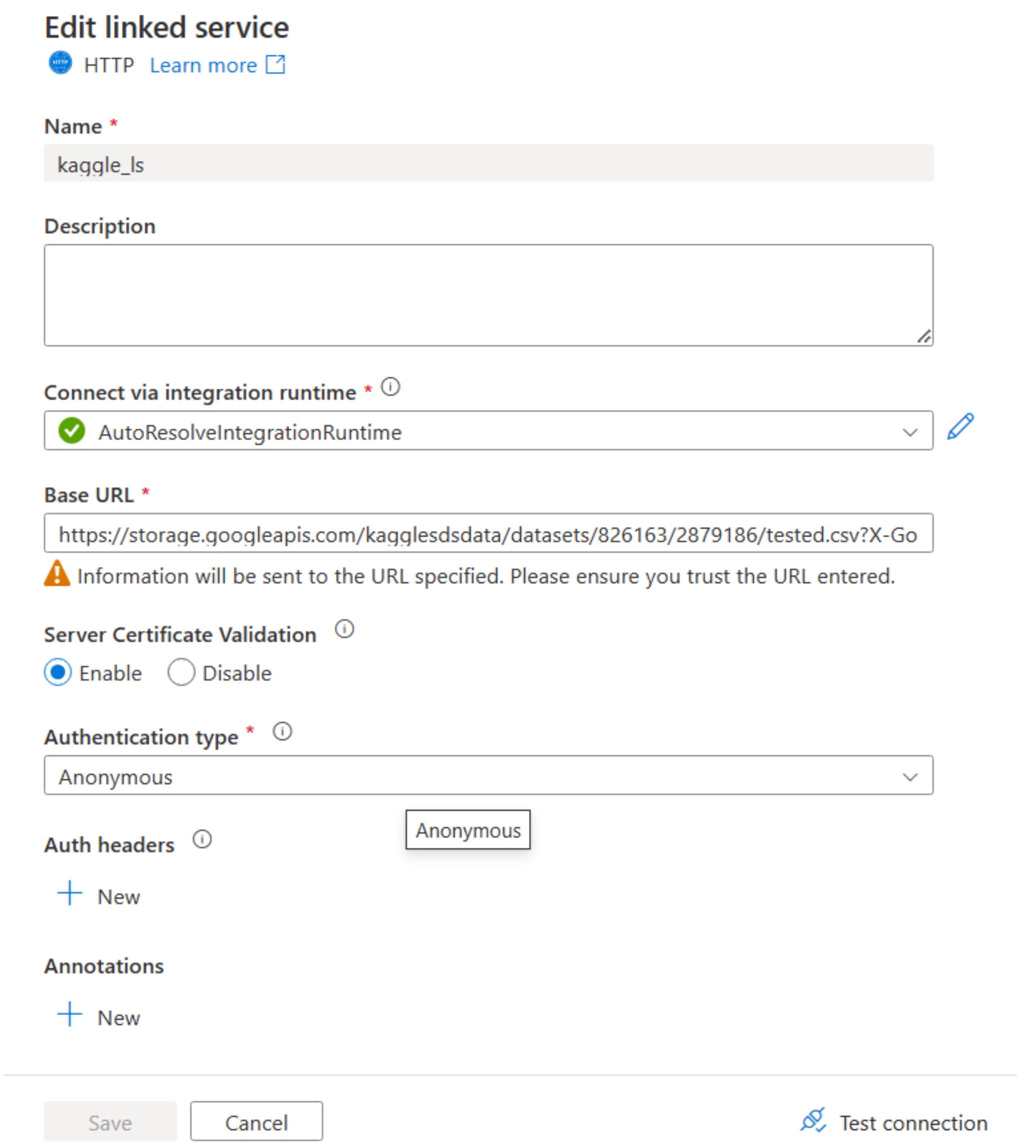
   Note - Sometimes, while copying, a link for the dataset through Kaggle is copied. To avoid this, download the dataset and then go to your downloads and then copy the link.

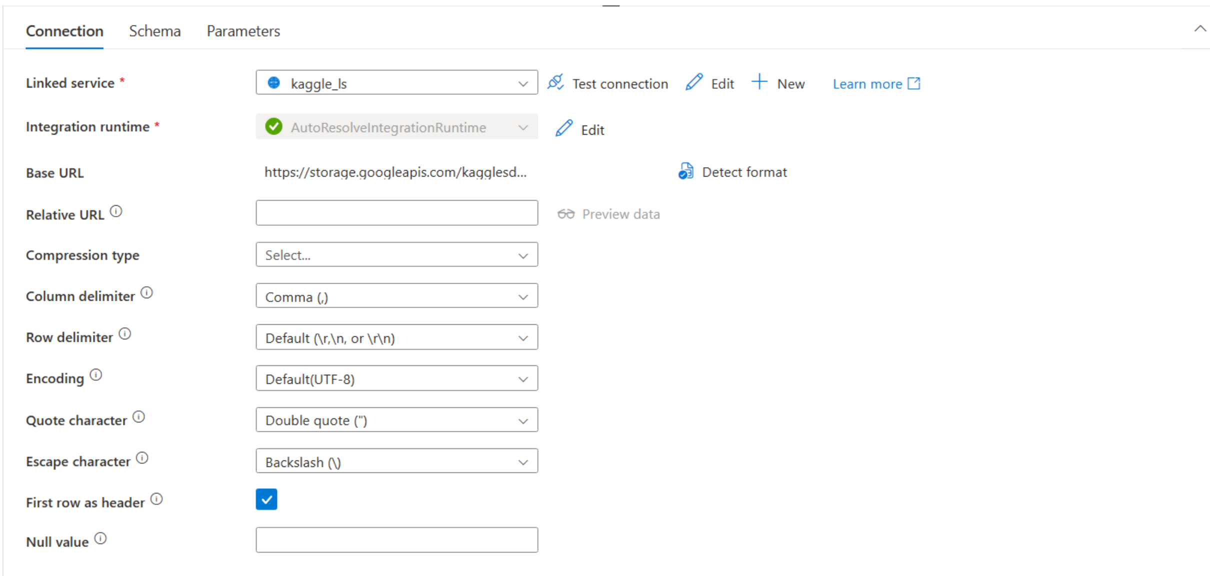
   The link should look something like this -

   https://storage.googleapis.com/kagglesdsdata/datasets/826163/2879186/tested.csv?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20230911%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20230911T095640Z&X-Goog-Expires=259200&X-Goog-SignedHeaders=host&X-Goog-Signature=6879e279ee27136ed2001d734890ce8fcf73c69e7d229d1b427e23a2a3c650b617042b96584cbbc122f25b4c0fbfal

4. Now for Authentication Type - Select Anonymous.
5. Final Linked service should look like this -

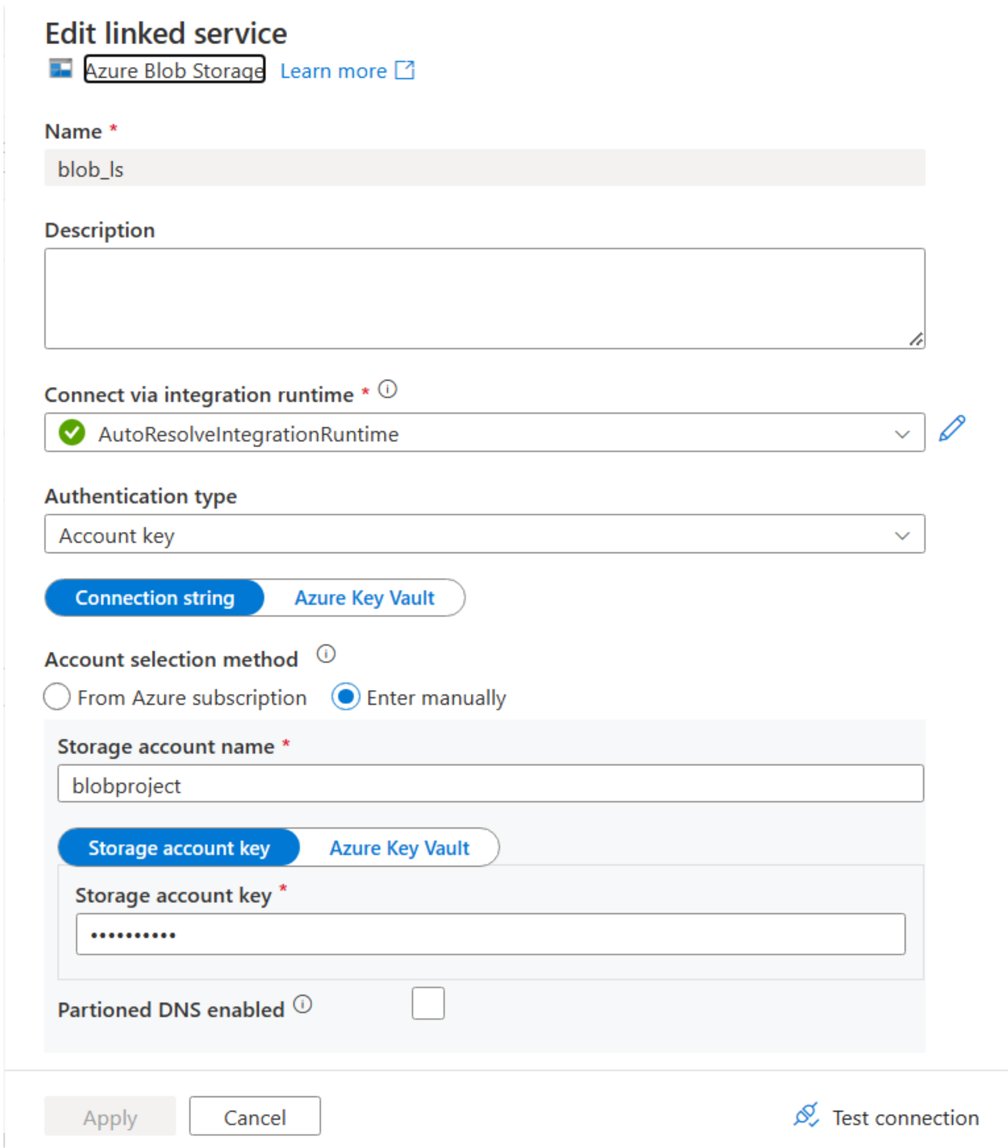


4. Final source dataset should look something like this -



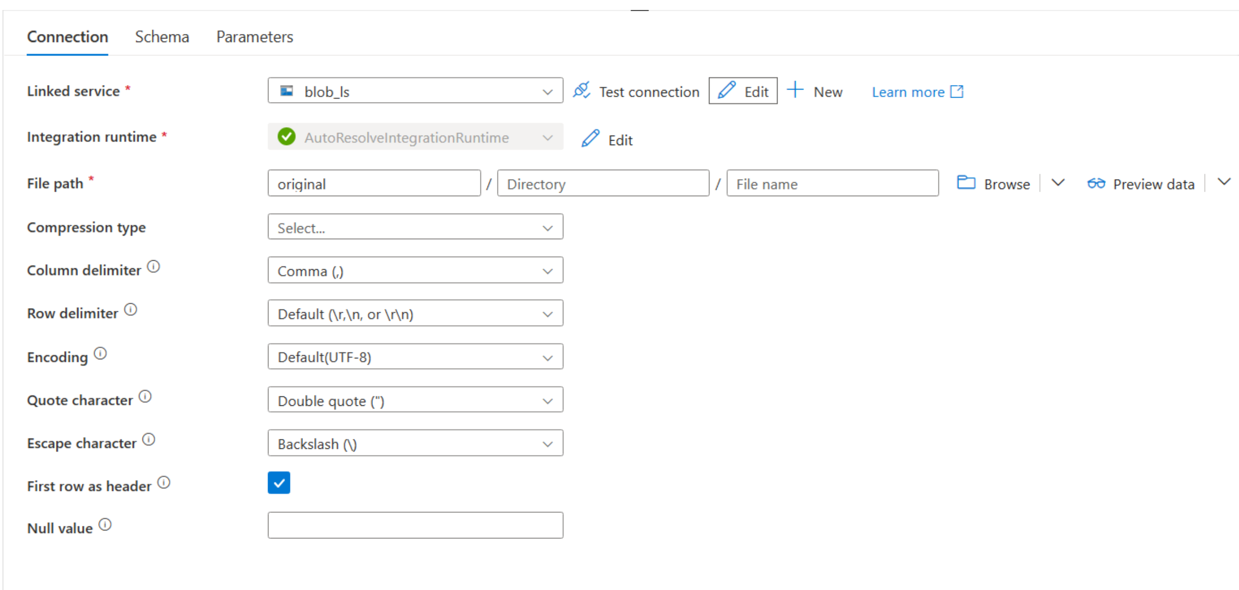
5. Similarly, configure the Sink dataset. For sink, Sink dataset → +New → Search for Azure Blob Storage → Select DelimitedText Format.

   Now similar to when we created Linked service for Source dataset, we do the same for Sink dataset.

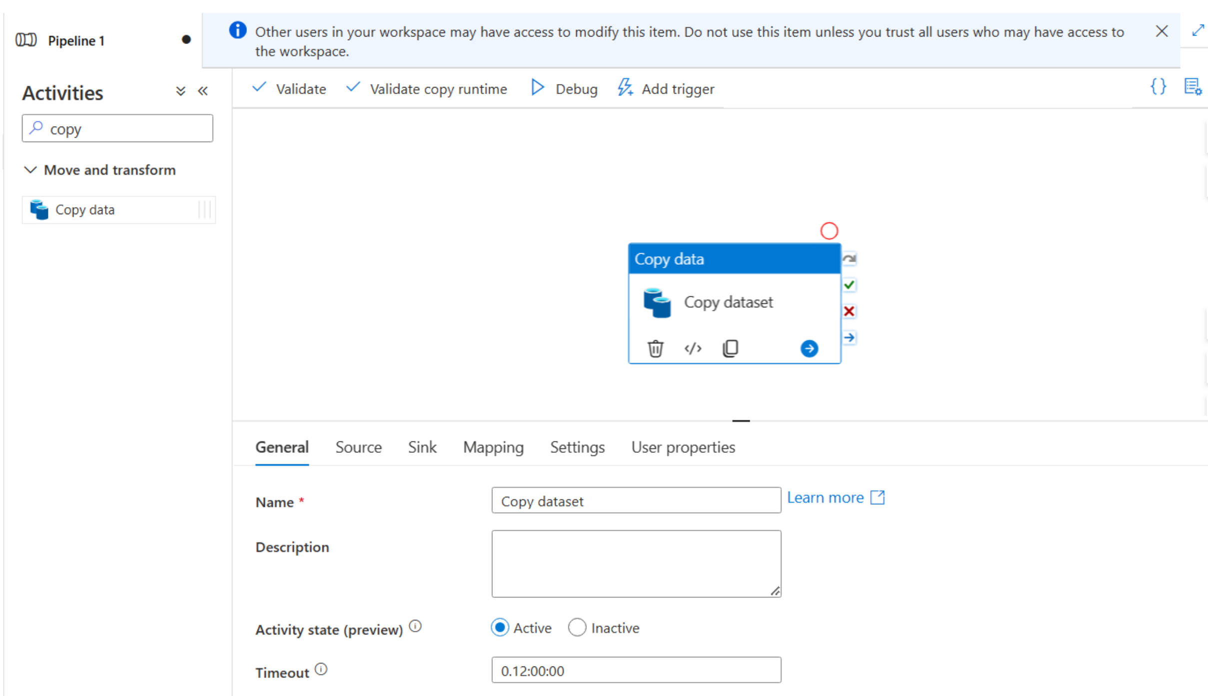


We get the Storage account name and Storage account key from when we configure the blob storage.

6. Final sink dataset should look something like this -

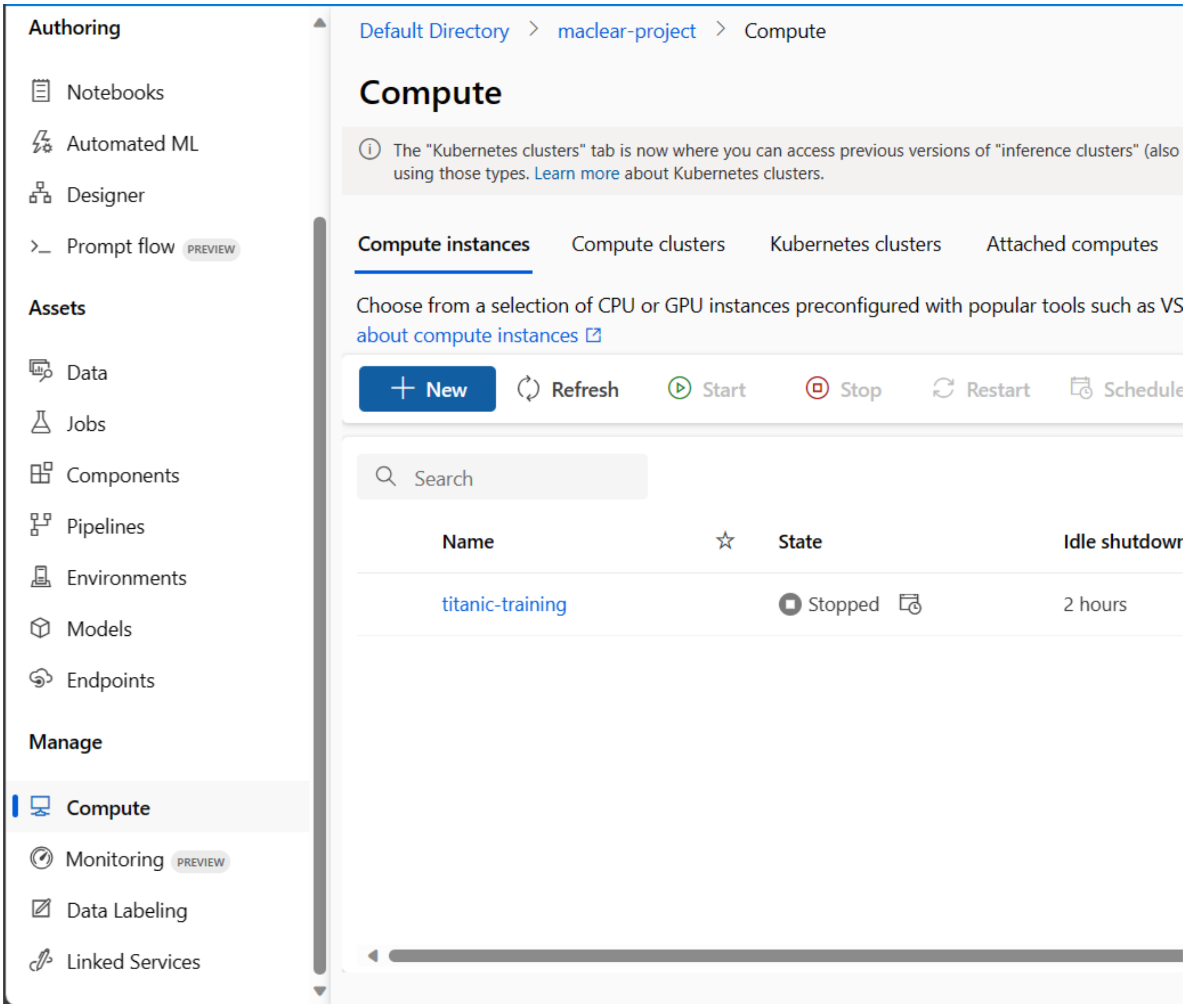


7. Finally, the Pipeline is configured. Now test the pipeline using the Debug or Add trigger option above the pipeline.



STEP 4: Creating Compute Instance in Azure ML Studio

1. Go to Compute → +New → And you will see this window:



2. Now type in any Compute name (preferably as per convention), Virtual Machine type as CPU and the Virtual Machine Size according to your needs (in this project we have used a Standard_DS11_v2 Virtual Machine).
3. Now, just click on Create. A Compute Instance is necessary as when we import our Auto ML Workspace in Azure Synapse Analytics, we need a Virtual Machine to run all algorithms.

STEP 5: Data Transformation

I .Configuring the first notebook

1. Now go to Activities and search for Notebook Activity -

   Now here, go to Settings tab → Notebook → +New → And you will see a Notebook open up.
2. Before configuring and adding code to the notebook, it is important to **Add a Spark Pool.**

   Go to Attach to → Manage pools → +New → and fill all fields accordingly:

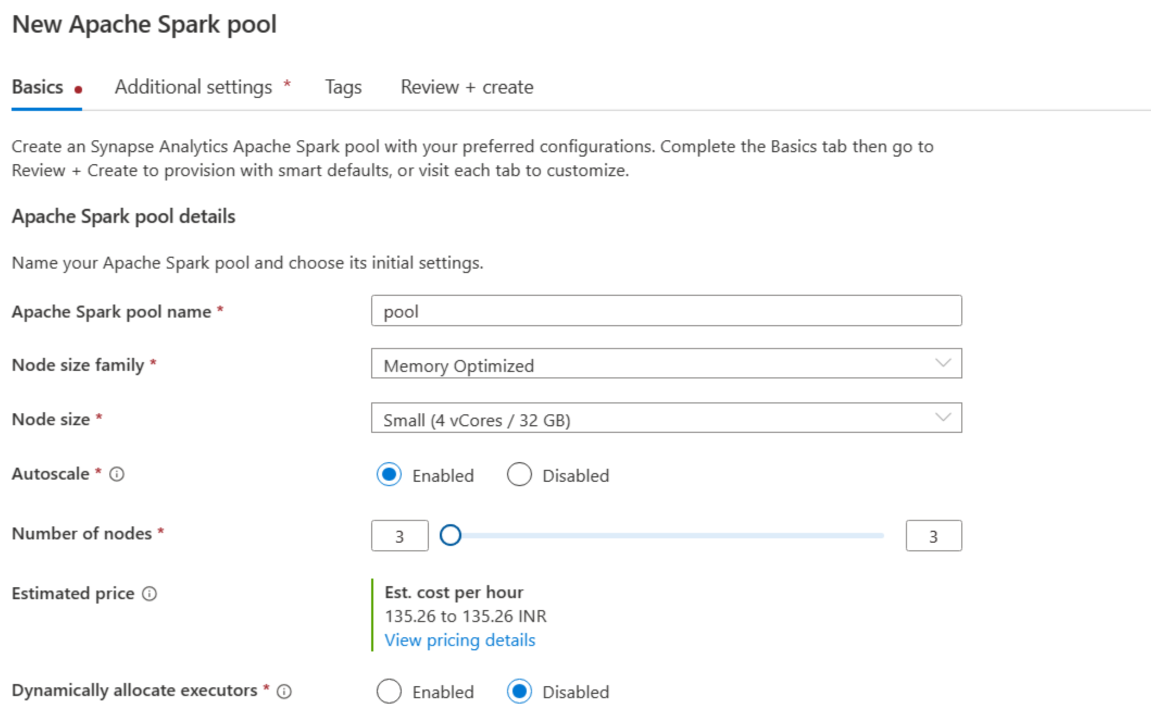
Apache Spark pool name: Any name as per convention

Node size family: Memory Optimized

Node size: Small (as per requirement)

Autoscale: Enabled

Number of nodes: 3



Also under Additional settings:

Leave all default setting as they are.

Just **set Apache Spark version to 2.4 (IMPORTANT)**

Now, just click on create and create a new Apache Spark.

3. Now, the notebook contains all code that is required to -
   a. Import our original dataset from the Blob Storage.
   b. Apply pre-processing steps to pre-process the given dataset.
   c. Split the dataset into Training and Testing datasets respectively.
   d. Connect Azure Automated Machine Learning to our notebook to train models on the Training dataset.
   e. Retrieve the model that has best accuracy and store it in Blob Storage.
   f. Also store the Test dataset into Blob Storage so that it can be referenced in the next notebook.

THE CODES ARE GIVEN BELOW -

1. **Import our original dataset from the Blob Storage.**

```
from datetime import datetime, timedelta
from azure.storage.blob import BlobServiceClient, generate_blob_sas, BlobSasPermissions
import pandas as pd

#enter credentials
account_name = 'blobproject'   #name of the storage account
account_key = 'Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ619xTur8S7RSX7UU1H2CzpgplUQUs+AStKf01PA=='   #got from access keys in the
container_name = 'original'   #name of container of original dataset

#create a client to interact with blob storage
connect_str = 'DefaultEndpointsProtocol=https;AccountName=' + account_name + ';AccountKey=' + account_key + ';EndpointSuffix=core.windo
blob_service_client = BlobServiceClient.from_connection_string(connect_str)

#use the client to connect to the container
container_client = blob_service_client.get_container_client(container_name)

#get a list of all blob files in the container
blob_list = []
for blob_i in container_client.list_blobs():
    blob_list.append(blob_i.name)



df_list = []
#generate a shared access signiture for files and load them into Python
for blob_i in blob_list:
    #generate a shared access signature for each blob file
    sas_i = generate_blob_sas(account_name = account_name,
                                container_name = container_name,
                                blob_name = blob_i,
                                account_key=account_key,
                                permission=BlobSasPermissions(read=True),
                                expiry=datetime.utcnow() + timedelta(hours=1))

    sas_url = 'https://' + account_name+'.blob.core.windows.net/' + container_name + '/' + blob_i + '?' + sas_i

    df = pd.read_csv(sas_url)
```

2. **Apply pre-processing steps to pre-process the given dataset.**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

df.isnull().sum()   #tells us the number of null values in each column of the csv
```

```
df.shape  #tells us the shape of the dataset
```

```
df['Age'].fillna(int(df['Age'].mean()), inplace=True)    #replacing all null values in age column with mean of all present ages
df['Fare'].fillna(int(df['Fare'].mean()), inplace=True)
df = df.reset_index(drop=True)    #resetting index of the dataset
df.isnull().sum()
```

```
y = df['Survived']              #extracting the predicted column from the dataset
X = df.drop(['Survived','Name','Ticket','Cabin'],axis=1)     #dropping all columns that do not have any effect on classification
X['Embarked'] = X["Embarked"].map({'Q':0, 'S':1, 'C':2}).astype(int)        #encoding or mapping categorical data
X['Sex'] = X["Sex"].map({'male':0, 'female':1}).astype(int)      #encoding or mapping categorical data
X["Age"] = X["Age"].astype(int)        #converting all ages and fares to integer values
X["Fare"] = X["Fare"].astype(int)
```

Final dataset looks like -

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	0	34	0	0	7	0
1	893	3	1	47	1	0	7	1
2	894	2	0	62	0	0	9	0
3	895	3	0	27	0	0	8	1
4	896	3	1	22	1	1	12	1
...	...	...	...	...	...	...	...	...
413	1305	3	0	30	0	0	8	1
414	1306	1	1	39	0	0	108	2
415	1307	3	0	38	0	0	7	1
416	1308	3	0	30	0	0	8	1
417	1309	3	0	30	1	1	22	2

418 rows × 8 columns

```
0      0
1      1
2      0
3      0
4      1
      ..
413    0
414    1
415    0
416    0
417    0
Name: Survived, Length: 418, dtype: int64
```

3. **Split the dataset into Training and Testing datasets respectively.**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_state = 1)    #splitting into training and testing d
frames=[X_train,y_train]
training_data = pd.concat(frames,axis=1, join="inner")
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
293	1185	1	0	53	1	1	81	1	0
58	950	3	0	30	1	0	16	1	0
197	1089	3	1	18	0	0	7	1	1
244	1136	3	0	30	1	2	23	1	0
189	1081	2	0	40	0	0	13	1	0
...	...	...	...	...	...	...	...	...	...
255	1147	3	0	30	0	0	7	1	0
72	964	3	1	29	0	0	7	1	1
396	1288	3	0	24	0	0	7	0	0
235	1127	3	0	20	0	0	7	1	0
37	929	3	1	21	0	0	8	1	1

355 rows × 9 columns

```
frames_test=[X_test,y_test]
test_data = pd.concat(frames_test,axis=1, join="inner")
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
358	1250	3	0	30	0	0	7	0	0
164	1056	2	0	41	0	0	13	1	0
17	909	3	0	21	0	0	7	2	0
67	959	1	0	47	0	0	42	1	0
4	896	3	1	22	1	1	12	1	1
...	...	...	...	...	...	...	...	...	...
171	1063	3	0	27	0	0	7	2	0
284	1176	3	1	2	1	1	20	1	1
295	1187	3	0	26	0	0	7	1	0
323	1215	1	0	33	0	0	26	1	0
122	1014	1	1	35	1	0	57	2	1

63 rows × 9 columns

4. **Connect Azure Automated Machine Learning to our notebook to train models on the Training dataset.**

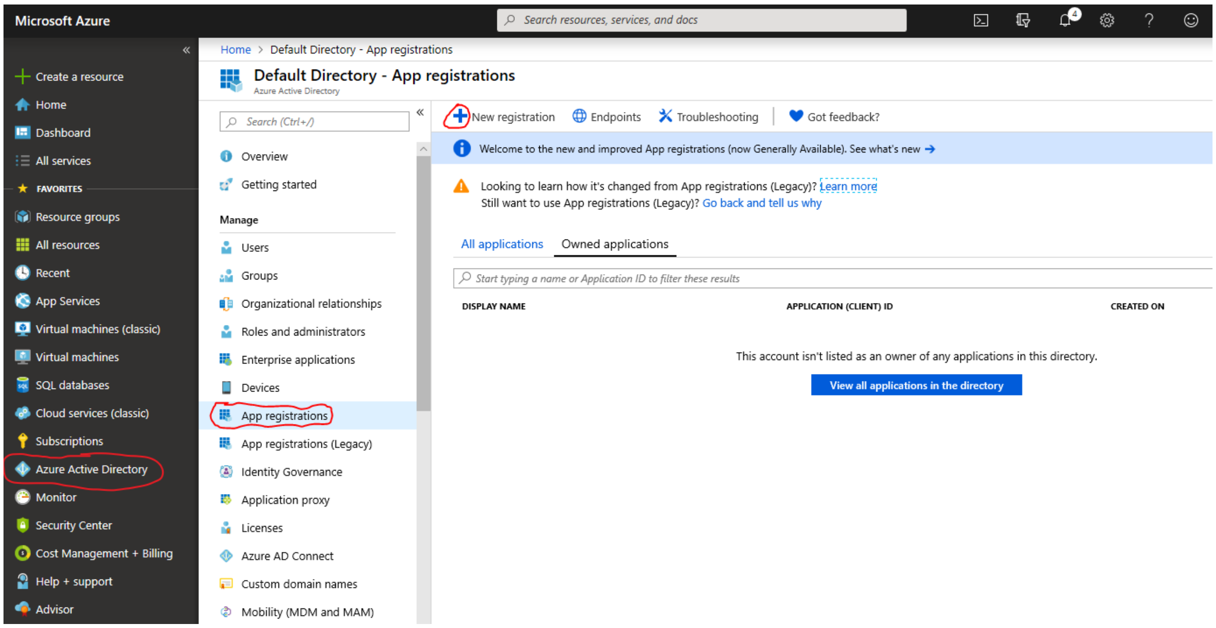
Service Principal Authentication

When setting up a machine learning workflow as an automated process, we recommend using Service Principal Authentication. This approach decouples the authentication from any specific user login, and allows managed access control.

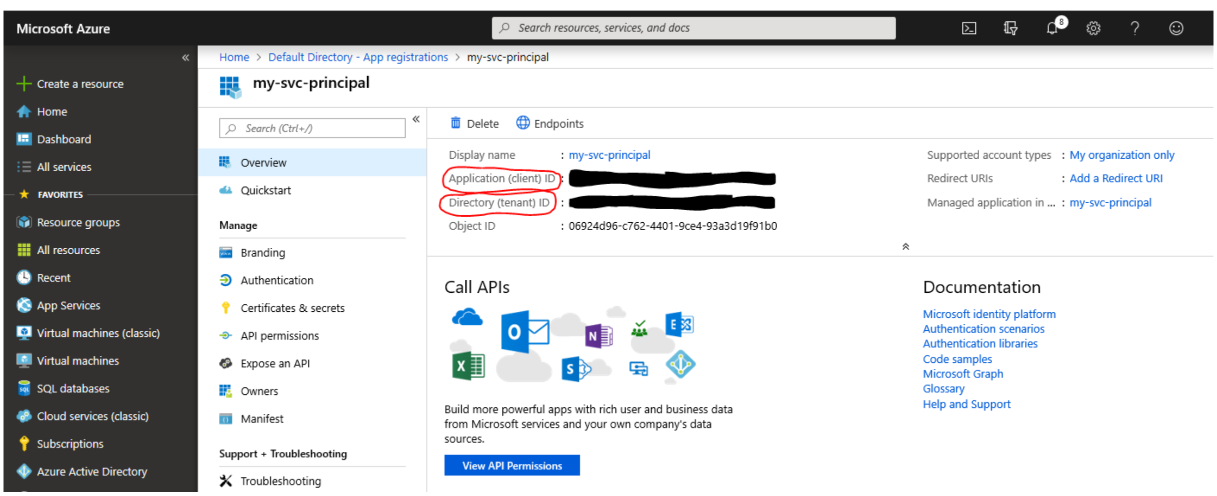
Note that you must have administrator privileges over the Azure subscription to complete these steps.

The first step is to create a service principal. First, go to Azure Portal, select **Azure Active Directory** and **App Registrations**. Then select **+New application**, give your service principal a name, for example *my-svc-principal*. You can leave other parameters as is.

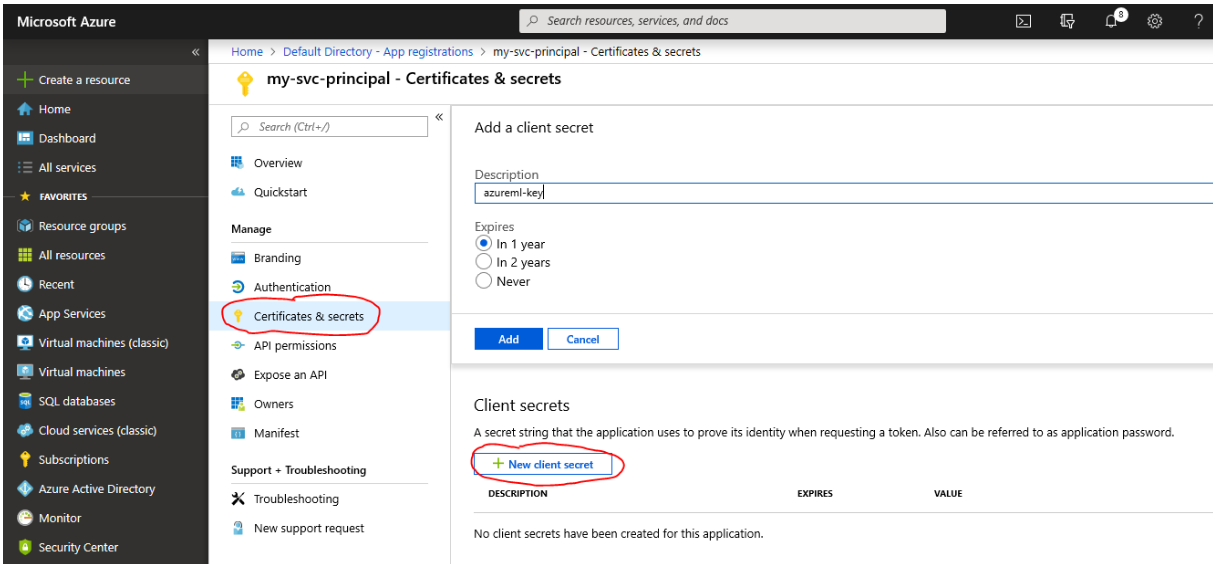
Then click **Register**.



From the page for your newly created service principal, copy the *Application ID* and *Tenant ID* as they are needed later.

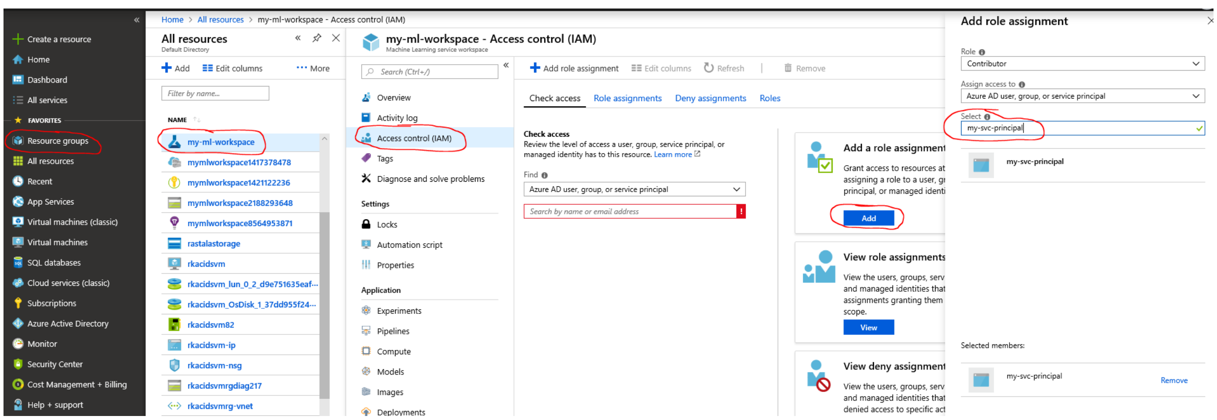


Then select **Certificates & secrets**, and **+New client secret** write a description for your key, and select duration. Then click **Add**, and copy the value of client secret to a secure location.



Finally, you need to give the service principal permissions to access your workspace. Navigate to **Resource Groups**, to the resource group for your Machine Learning Workspace.

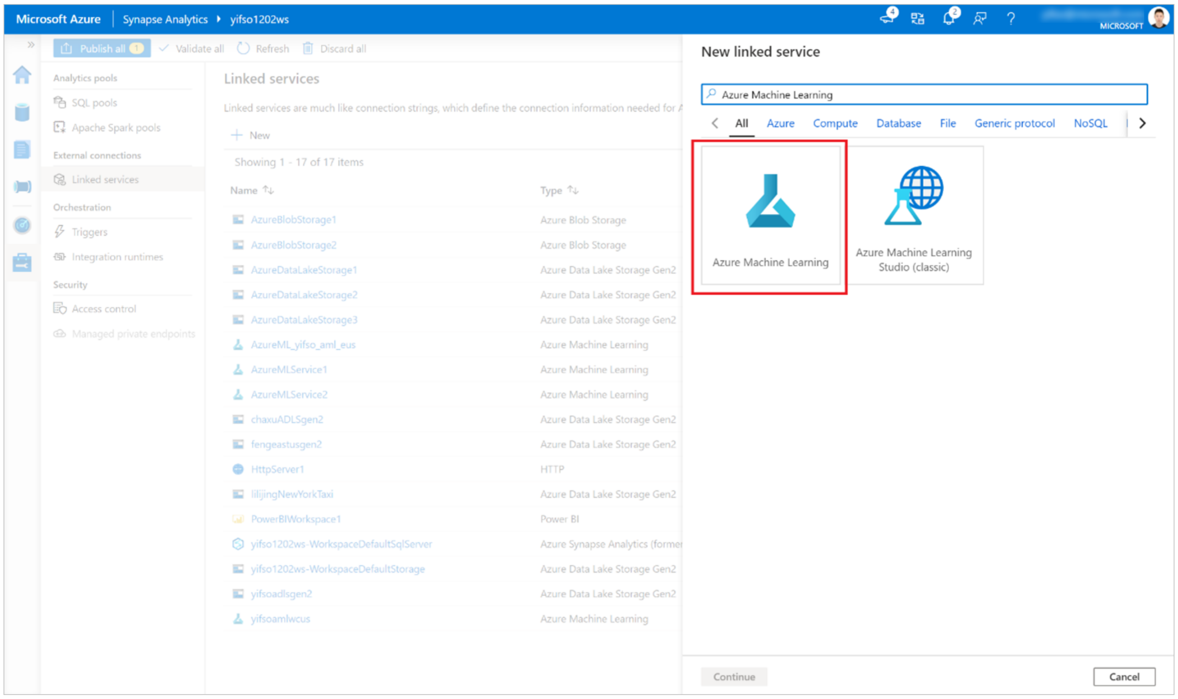
Then select **Access Control (IAM)** and **Add a role assignment**. For *Role*, specify which level of access you need to grant, for example *Contributor*. Start entering your service principal name and once it is found, select it, and click **Save**.



Now you are ready to use the service principal authentication. For example, to connect to your Workspace, see code below and enter your own values for tenant ID, application ID, subscription ID, resource group and workspace.

Create an Azure ML linked service

1. In the Synapse workspace where you want to create the new Azure Machine Learning linked service, go to **Manage** -> **Linked services**, create a new linked service with type "Azure Machine Learning".



2. Fill out the form:

- Provide the details about the Azure Machine Learning workspace you want to link to. This includes details about subscription and workspace name.
- Select Authentication Method: **Service Principal**
- Service principal ID: This is the **application (client) ID** of the Application.

```
import os
from azureml.core import Workspace
from azureml.core.authenticationimport ServicePrincipalAuthentication


svc_pr= ServicePrincipalAuthentication(
    tenant_id="my-tenant-id",              #copied earlier along with application id
    service_principal_id="my-application-id",
    service_principal_password="secret-value")


ws= Workspace(
    subscription_id="my-subscription-id",
    resource_group="my-ml-rg",
    workspace_name="my-ml-workspace",
    auth=svc_pr
    )
```

```
import logging

automl_settings = {
    "iteration_timeout_minutes": 10,
    "experiment_timeout_minutes": 30,   #depends on your dataset
    "enable_early_stopping": True,
    "primary_metric": 'accuracy',   #as classification
    "featurization": 'auto',
    "verbosity": logging.INFO,
    "n_cross_validations": 2}
```

```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(task='classification',
                             debug_log='automated_ml_errors.log',
                             training_data = training_data,
                             model_explainability = False,
                             label_column_name ="Survived",**automl_settings)   #as survived column is the target column
```

```
from azureml.core.experiment import Experiment

# Start an experiment in Azure Machine Learning
experiment = Experiment(ws, "aml-titanic-project")
# tags = {"Titanic": "classification"}
local_run = experiment.submit(automl_config, show_output=True)

# Use the get_details function to retrieve the detailed output for the run.
run_details = local_run.get_details()    #starting training of all models
```

The output looks something like this-

```
...
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
****************************************************************************************************

 ITERATION   PIPELINE                                       DURATION      METRIC      BEST
         0   MaxAbsScaler LightGBM                          0:00:15       1.0000    1.0000
         1   MaxAbsScaler XGBoostClassifier                 0:00:14       1.0000    1.0000
         2   MaxAbsScaler ExtremeRandomTrees                0:00:14       1.0000    1.0000
         3   SparseNormalizer XGBoostClassifier             0:00:13       1.0000    1.0000
         4   MaxAbsScaler LightGBM                          0:00:15       1.0000    1.0000
         5   MaxAbsScaler LightGBM                          0:00:14       0.8164    1.0000
         6   StandardScalerWrapper XGBoostClassifier        0:00:14       1.0000    1.0000
         7   MaxAbsScaler LogisticRegression                0:00:13       1.0000    1.0000
         8   StandardScalerWrapper ExtremeRandomTrees       0:00:14       0.8927    1.0000
         9   StandardScalerWrapper XGBoostClassifier        0:00:14       1.0000    1.0000
        10   SparseNormalizer LightGBM                      0:00:13       1.0000    1.0000
        11   StandardScalerWrapper XGBoostClassifier        0:00:10       1.0000    1.0000
        12   MaxAbsScaler LogisticRegression                0:00:14       1.0000    1.0000
        13   MaxAbsScaler SGD                               0:00:14       1.0000    1.0000
        14   StandardScalerWrapper XGBoostClassifier        0:00:13       1.0000    1.0000
        15   SparseNormalizer RandomForest                  0:00:14       1.0000    1.0000
```

5. **Retrieve the model that has best accuracy and store it in Blob Storage.**

```
# Get best model
best_run, fitted_model = local_run.get_output()
```

```
import pickle
pickle.dump(fitted_model, open('model.pkl', 'wb'))    #saving the model as model.pkl
```

```
from azure.storage.blob import BlobServiceClient
storage_account_key ="Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ619xTur8S7RSX7UU1H2CzpgplUQUs+AStKf01PA=="
storage_account_name ="blobproject"
connection_string ="DefaultEndpointsProtocol=https;AccountName=blobproject;AccountKey=Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ61
container_name ="picklefile"   #name of container

def uploadToBlobStorage (file_path, file_name):
    blob_service_client = BlobServiceClient.from_connection_string (connection_string)
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=file_name)

    with open(file_path, "rb") as data:
        blob_client.upload_blob (data)
```

```
uploadToBlobStorage('model.pkl', 'modelNew.pkl')    #saving the model in blob storage as modelNew.pkl
```

6. **Store the Test dataset into Blob Storage so that it can be referenced in the next notebook.**

```
test_data.to_csv('TestData.csv')   #converting dataframe to csv file
```

```
from azure.storage.blob import BlobServiceClient
storage_account_key ="Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ619xTur8S7RSX7UU1H2CzpgplUQUs+AStKf01PA=="
storage_account_name ="blobproject"
connection_string ="DefaultEndpointsProtocol=https;AccountName=blobproject;AccountKey=Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ61
container_name ="testdata"

def uploadToBlobStorage (file_path, file_name):
    blob_service_client = BlobServiceClient.from_connection_string (connection_string)
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=file_name)

    with open(file_path, "rb") as data:
        blob_client.upload_blob (data)
```

```
uploadToBlobStorage('TestData.csv', 'TestDataNew.csv')    #storing csv file by the name TestDataNew.csv
```

II.Configuring the second notebook

1. Now, the notebook contains all code that is required to -
    a. Import our Test dataset from blob storage.
    b. Import our model with best accuracy from blob storage.
    c. Clean our Test dataset and make it in the format we need.
    d. Test best model accuracy on the Test dataset.
    e. Print the accuracy using accuracy_score metric.
    f. Convert the Predicted values to a data frame and also the original Testing values to a data frame.
    g. Combine all resulting data frames to make a final Dataset with original and predicted values.
    h. Store this Final dataset in the blob storage.

THE CODES ARE GIVEN BELOW -

1. **Import our Test dataset from blob storage.**

```
from datetime import datetime, timedelta
from azure.storage.blob import BlobServiceClient, generate_blob_sas, BlobSasPermissions
import pandas as pd

#enter credentials
account_name = 'blobproject'
account_key = 'Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ619xTur8S7RSX7UU1H2CzpgplUQUs+AStKf01PA=='
container_name = 'testdata'

#create a client to interact with blob storage
connect_str = 'DefaultEndpointsProtocol=https;AccountName=' + account_name + ';AccountKey=' + account_key + ';EndpointSuffix=core.windo
blob_service_client = BlobServiceClient.from_connection_string(connect_str)

#use the client to connect to the container
container_client = blob_service_client.get_container_client(container_name)

#get a list of all blob files in the container
blob_list = []
for blob_i in container_client.list_blobs():
    blob_list.append(blob_i.name)



df_list = []
#generate a shared access signiture for files and load them into Python
for blob_i in blob_list:
    #generate a shared access signature for each blob file
    sas_i = generate_blob_sas(account_name = account_name,
                                container_name = container_name,
                                blob_name = blob_i,
                                account_key=account_key,
                                permission=BlobSasPermissions(read=True),
                                expiry=datetime.utcnow() + timedelta(hours=1))

    sas_url = 'https://' + account_name+'.blob.core.windows.net/' + container_name + '/' + blob_i + '?' + sas_i

    df = pd.read_csv(sas_url)       #we use the sas token method to retreive the csv file
```

2. **Import our model with best accuracy from blob storage.**

```
import pickle
from azure.storage.blob import BlobServiceClient

# Azure Blob Storage connection information
connection_string = "DefaultEndpointsProtocol=https;AccountName=blobproject;AccountKey=Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ6
container_name = "picklefile"
blob_name = "modelNew.pkl"  # Replace with the name of your pickle file

# Create a BlobServiceClient using the connection string
blob_service_client = BlobServiceClient.from_connection_string(connection_string)

# Get the blob from the container
blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)

# Download the blob's content as bytes
blob_data = blob_client.download_blob()
pickle_bytes = blob_data.readall()

# Load the pickle data into a Python object
loaded_data = pickle.loads(pickle_bytes)

# Now, you can use the loaded_data in your Python code
```

```
loaded_data
```

3. **Clean our Test dataset and make it in the format we need.**

```
df.shape
```

```
X = df.drop(df.columns[[0]], axis=1)
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
0	1250	3	0	30	0	0	7	0	0
1	1056	2	0	41	0	0	13	1	0
2	909	3	0	21	0	0	7	2	0
3	959	1	0	47	0	0	42	1	0
4	896	3	1	22	1	1	12	1	1
...	...	...	...	...	...	...	...	...	...
58	1063	3	0	27	0	0	7	2	0
59	1176	3	1	2	1	1	20	1	1
60	1187	3	0	26	0	0	7	1	0
61	1215	1	0	33	0	0	26	1	0
62	1014	1	1	35	1	0	57	2	1

63 rows × 9 columns

```
for blob in blob_service_client.get_container_client(container_name).list_blobs():
    print(blob.name)
```

```
y_testing = X.pop("Survived").to_frame()
```

4. **Test best model accuracy on the Test dataset.**

```
y_predict = loaded_data.predict(X)  #here loaded_data is our model
```

5. **Print the accuracy using accuracy_score metric.**

```
from sklearn.metrics import accuracy_score
import numpy as np

# Calculate root-mean-square error
y_actual = y_testing.values.flatten().tolist()
y_actual=np.array(y_actual)
acc = accuracy_score(y_actual, y_predict)    #accuracy_score metric is used
```

```
print("Accuracy: ")
print(acc)
```

6. **Convert the Predicted values to a data frame and also the original Testing values to a data frame.**

```
y_predict     #see shape of y_predict
```

```
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1])
```

```
y_actual
```

```
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1])
```

```
df_predict = pd.DataFrame(y_predict.T, columns=['Survived_Predicted'])
```

	Survived_Predicted
0	0
1	0
2	0
3	0
4	1
...	...
58	0
59	1
60	0
61	0
62	1

63 rows × 1 columns

```
df_actual = pd.DataFrame(y_actual.T, columns=['Survived_Actual'])
```

	Survived_Actual
0	0
1	0
2	0
3	0
4	1
...	...
58	0
59	1
60	0
61	0
62	1

63 rows × 1 columns

7. **Combine all resulting data frames to make a final Dataset with original and predicted values.**

```
frames=[X,df_actual,df_predict]
df_final = pd.concat(frames,axis=1, join="inner")
df_final
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived_Actual	Survived_Predicted
0	1250	3	0	30	0	0	7	0	0	0
1	1056	2	0	41	0	0	13	1	0	0
2	909	3	0	21	0	0	7	2	0	0
3	959	1	0	47	0	0	42	1	0	0
4	896	3	1	22	1	1	12	1	1	1
...	...	...	...	...	...	...	...	...	...	...
58	1063	3	0	27	0	0	7	2	0	0
59	1176	3	1	2	1	1	20	1	1	1
60	1187	3	0	26	0	0	7	1	0	0
61	1215	1	0	33	0	0	26	1	0	0
62	1014	1	1	35	1	0	57	2	1	1

63 rows × 10 columns

8. **Store this Final dataset in the blob storage.**

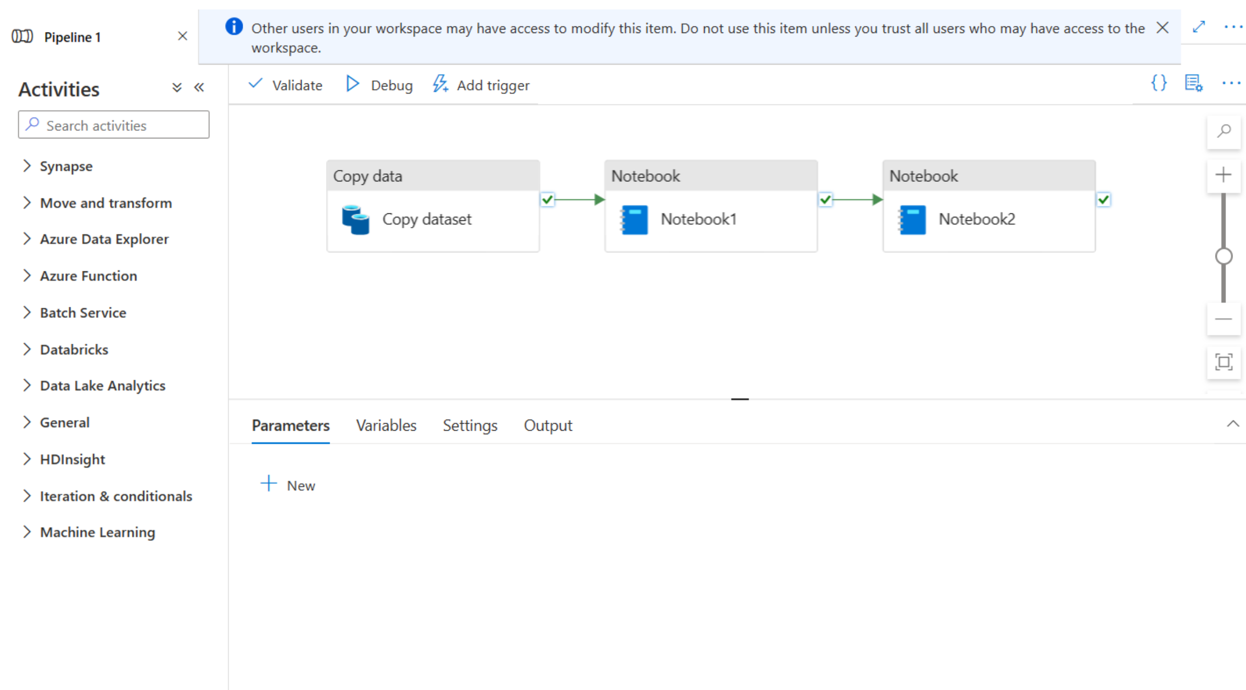
```
df_final.to_csv('FinalData.csv')
```

```
from azure.storage.blob import BlobServiceClient
storage_account_key ="Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ619xTur8S7RSX7UU1H2CzpgplUQUs+AStKf01PA=="
storage_account_name ="blobproject"
connection_string ="DefaultEndpointsProtocol=https;AccountName=blobproject;AccountKey=Iyqf1hJVmyb9jF9J0iUJS9DBdO//TAjJD3SZIvD5uB1tmkZ61
container_name ="final"

def uploadToBlobStorage (file_path, file_name):
    blob_service_client = BlobServiceClient.from_connection_string (connection_string)
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=file_name)

    with open(file_path, "rb") as data:
        blob_client.upload_blob (data)
```

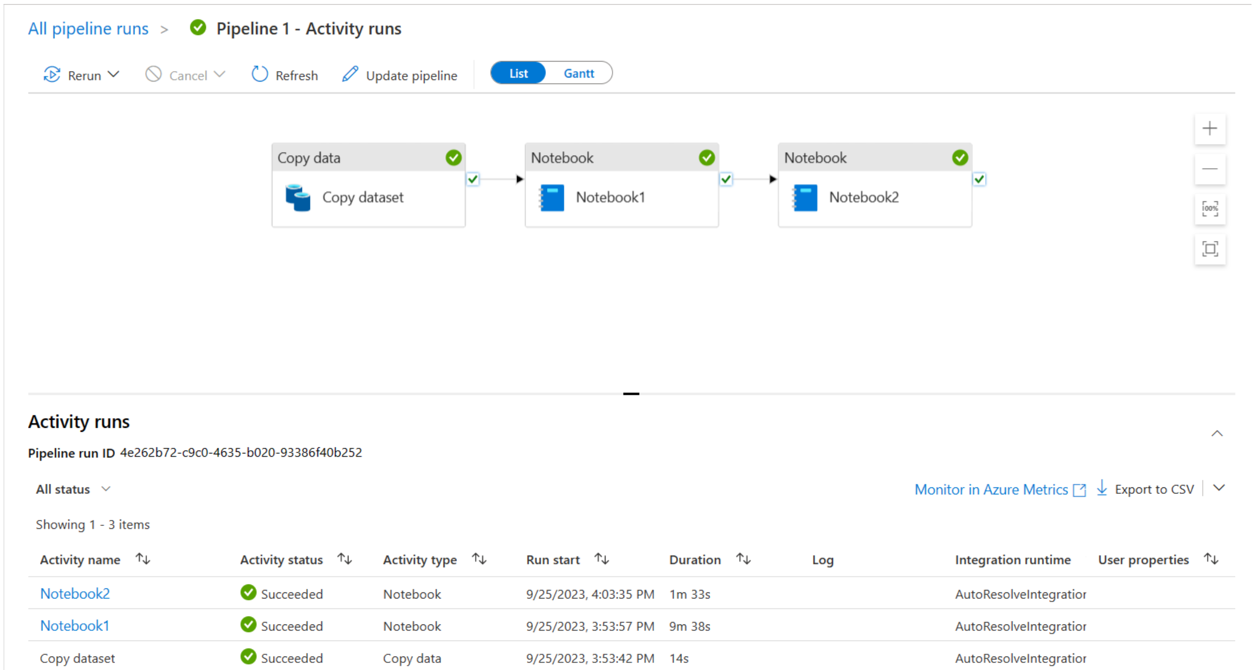
```
uploadToBlobStorage('FinalData.csv', 'FinalDataNew.csv')
```

STEP 6: Running the Pipeline

Now, the pipeline is configured. It looks something like this -



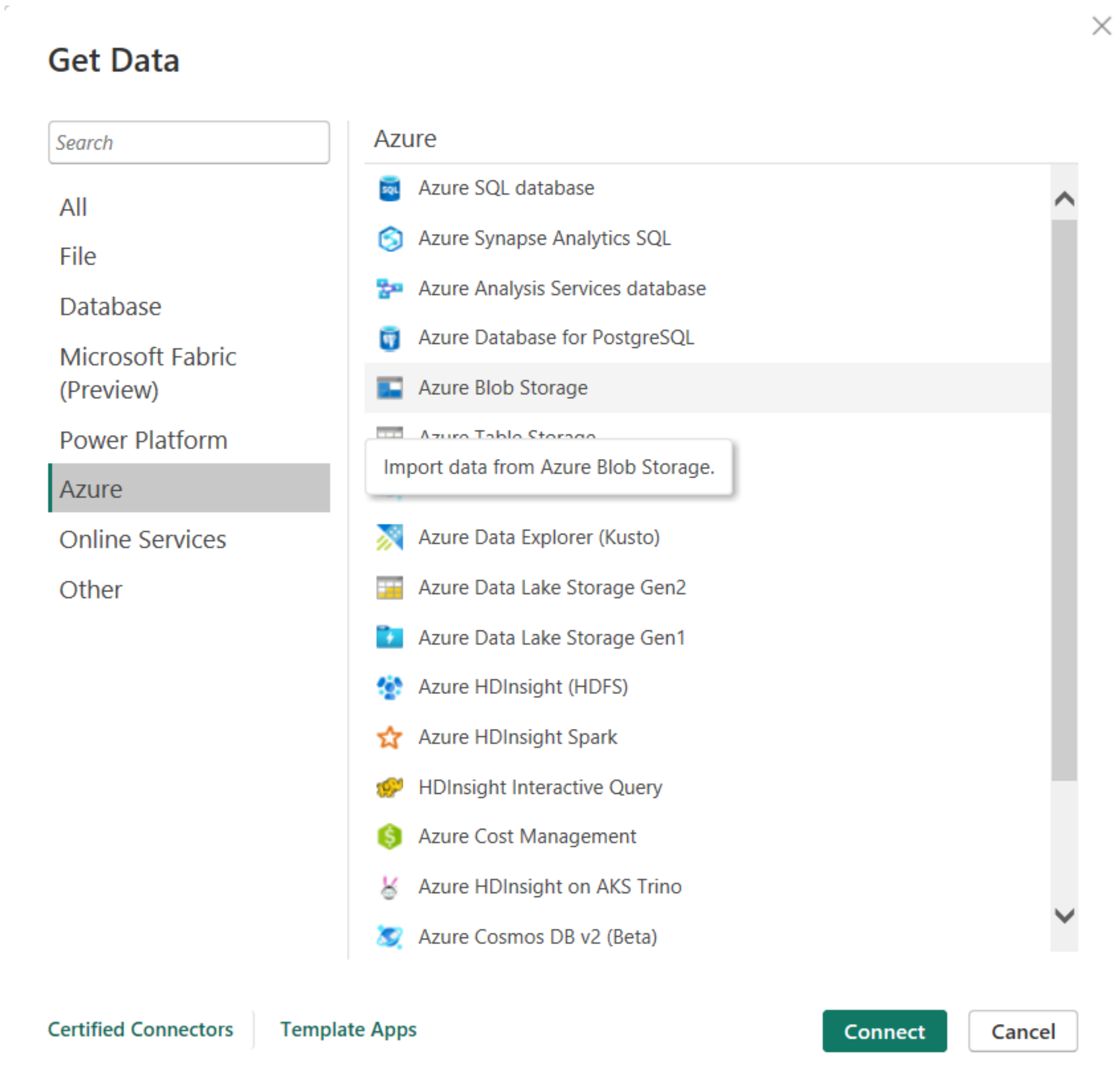
Publish all pending updates and run the new Pipeline using the Debug or Trigger Now options.



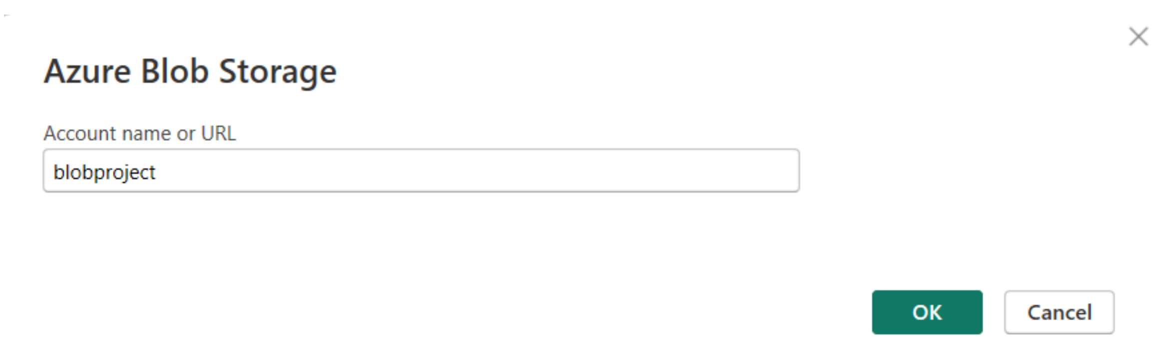
STEP 7: Data Reporting

To export all the Final Database from the blob storage and to prepare reports on the same, we use POWER BI.

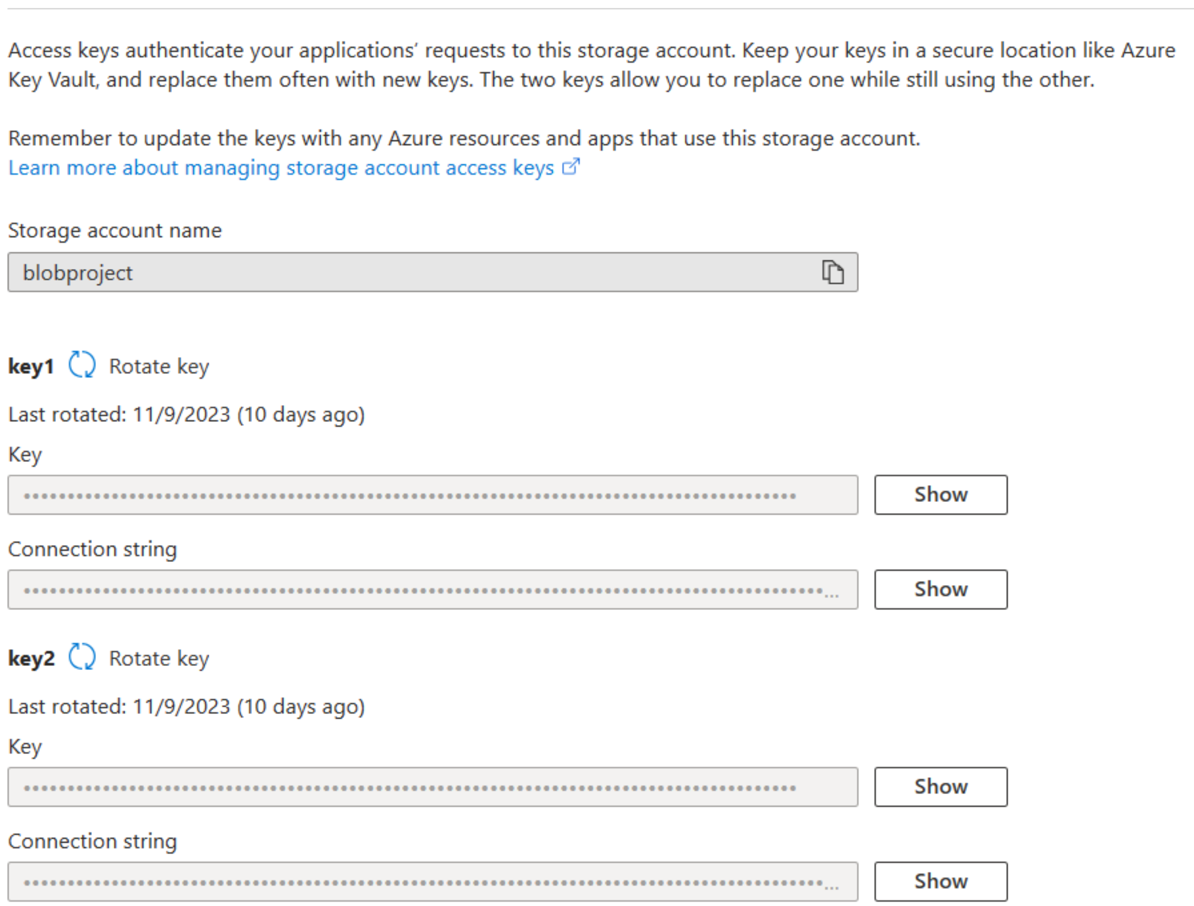
Use the Get Data tab → More…. → Azure → Azure Blob Storage, as shown below:



After clicking on Connect option, it will ask you the name of the Account. Here, you have to type in the name of your Blob Storage account which in this case is - blobproject.

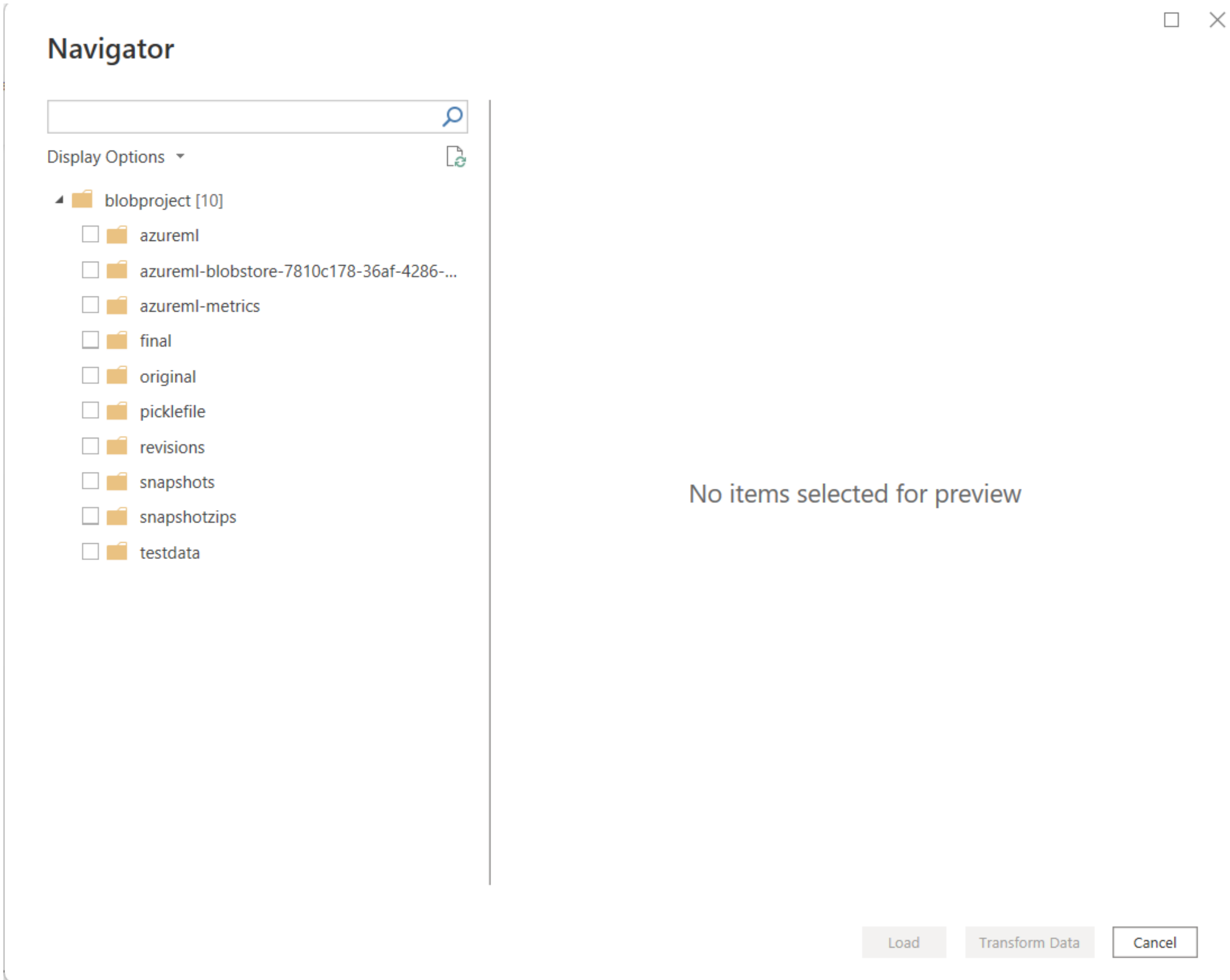


For Account key, copy paste one of the Access Keys of the Storage.

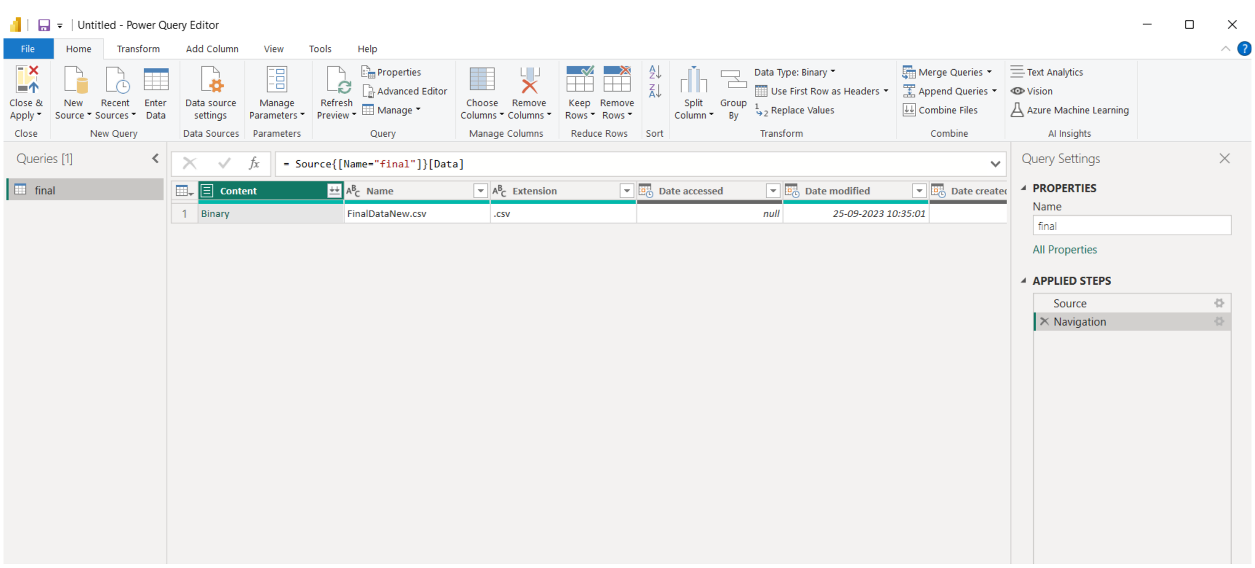


Click on Connect to connect.

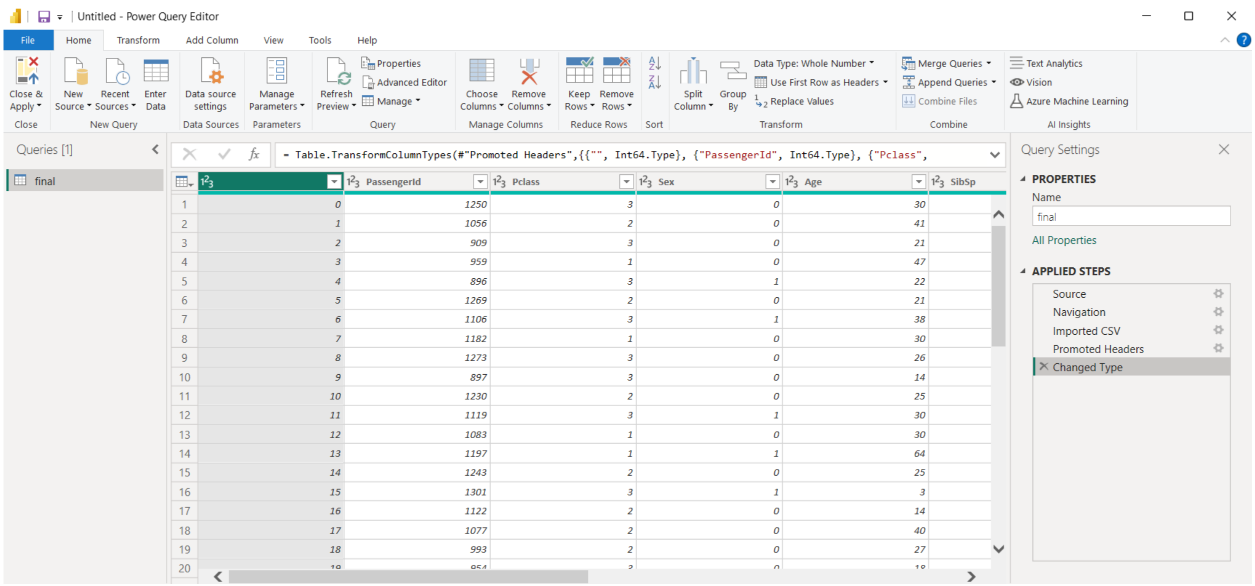
Now a Navigator window will open and here, select the container in which your final dataset is stored.



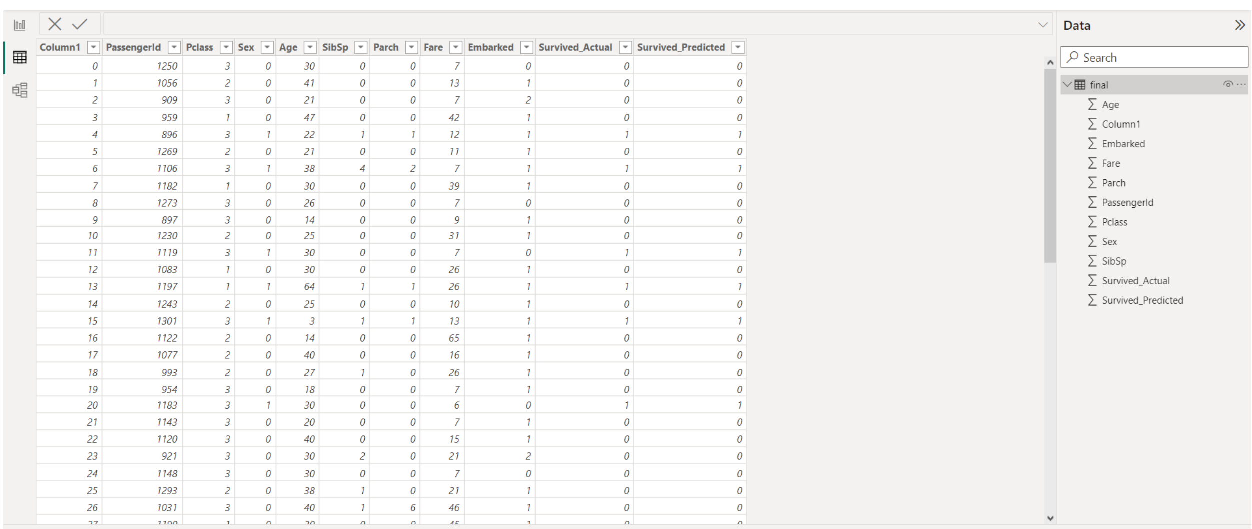
Here, as the Final Dataset is in the final container of the blob storage, click on final → Transform Data. The following screen opens.



Now click on **Binary** (under the Content column). This opens up the following window →



Here, now just click on Close and Apply in the top left to aply these changes and import the csv. This loads the csv file in our POWER BI desktop.



THIS IS HOW OUR FINAL DATASET LOOKS LIKE.

Now use all POWER BI tools to build an informative report.