

Databricks, Spark, Machine Learning and Azure Synapse Analytics

AN END-TO-END EXAMPLE OF DATA IN THE CLOUD

KINGABY, SIMON

Table of Contents

Summary	3
About the Author	3
Setup	4
Tools to Download and Install	4
Create some Azure Resources	4
Resource Group	4
Data Factory	4
Blob Storage	4
Data Lake	5
Key Vault	5
Application Insights	6
SQL Server	6
Databricks	6
Container Registry	7
Machine Learning	7
Exercise 1: Getting Some Data to Analyze	9
Exercise 2: Loading the Data Lake	11
Step 1: Upload the file to Blob storage	11
Step 2: Use Data Factory to Transfer the Blob to the Data Lake	12
Exercise 3: Processing the data in Databricks	13
Connecting to the Data Lake	14
Loading the Parquet File	15
Exercise 4: Creating the Machine Learning Model	16
Exercise 5: Detour Ahead! Create a Custom Docker Base Image	18
Install Docker Desktop on Your Local Machine	18
Clone and Modify a New Docker Image	18
To Edit the Docker Image	19
Exercise 6: Configure the Model for Deployment	20
Exercise 7: Build and Deploy the Docker image	23
Exercise 8: Testing the Webservice	24
A couple of things to know	26
Exercise 9: Loading the Data Warehouse	27

Create the Data Warehouse 27

Create the Databricks ETL Script..... 28

Connect to the DW and Create the Target Tables..... 30

Back to Data Factory to Load the DW 31

Exercise 10: Creating a Power BI Report..... 32

Databricks, Spark, Machine Learning and Azure Synapse Analytics

An End-To-End Example of Data in The Cloud

Summary

You've heard about Azure Data Lake and Azure Data Warehouse, now called Azure Synapse Analytics. You've also heard about Azure Data Factory and Azure Data Bricks. You might even have heard about Python, Spark, and Azure Machine Learning. In this fast-paced, all-demo session, we will walk through the process of ingesting data into the Data Lake with Data Factory, analyzing it in Databricks with Spark and Machine Learning, outputting the resulting model as a Docker image, running the model against a prediction dataset and saving it to the Data Warehouse and reporting on it in Power BI. You will walk away with working code and an overall understanding of how all these tools can help you develop advanced analytics solutions in the modern data landscape.

About the Author

Simon Kingaby has been wrangling data for 20 plus years. For the past 5 years he's been moving data to the cloud from DB2, Oracle, Netezza and SQL Server. Now on the Big Data team at Deloitte Global, he's moving data from on-prem data sources up to a Data Lake and then slice and dicing it through Databricks into a Machine Learning model and on to the Data Warehouse where the data is presented in Power BI. Simon also teaches the Data Analysis and Visualization Bootcamp at Vanderbilt University, has published several Alexa skills, and regularly speaks at conferences.

Setup

Before working with the exercises in this document, we need to set up the Azure environment.

If you do not have an Azure account, sign up for one. (<https://azure.microsoft.com/en-us/free/>)

If you have an MSDN subscription, you can use the Azure account associated with it.

Many of the resources we will create have an associated charge for use. The worst is the Data Warehouse which has an hourly rate just for existing. While the total cost of completing these exercises should be less than \$50, this is not guaranteed. If nothing else, you will definitely want to delete the DW instance when you are done.

Tools to Download and Install

In order to complete the exercises, you will need to upload a file to the Data Lake, this requires the Microsoft Azure Storage Explorer.

- <https://azure.microsoft.com/en-us/features/storage-explorer/>

Create some Azure Resources

First things first, you will want all the exercise created items to be in a single, new Resource Group so that you can delete the Resource Group when you are done (and stop the charges against your Azure account). You will also want to create several resources, detailed below, in the new Resource Group.

Resource Group

1. Create a new Resource Group to put all the exercise work in.
 - a. In the Azure Portal, click on Resource Groups on the sidebar.
 - b. Click the Add button.
 - c. Give it a name in the Resource group box and click Review + Create at the bottom.
 - d. Then click Create at the bottom.

Data Factory

2. Create a Data Factory
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Data Factory
 - d. Select the Data Factory option
 - e. Click Create
 - f. Enter a name for your data factory
 - g. Select the right subscription
 - h. Pick the resource group you created above
 - i. Select a Location
 - j. Uncheck the Enable GIT box
 - k. Click Create

Blob Storage

3. Create a Blob Storage Account in the **North Central US** region
 - a. Open the Resource Group you created above.

- b. Click the Add button
- c. Search for Storage
- d. Pick Storage Account
- e. Click Create
- f. Select the right subscription
- g. Pick the resource group you created above
- h. Enter a Storage account name
- i. Select the **North Central US** Location (this is necessary for the ML exercise)
- j. Select Standard Performance
- k. Select Account kind: BlobStorage
- l. Select Locally-redundant Storage (LRS)
- m. Select the Cool Access tier
- n. Click Review + Create
- o. Click Create

Data Lake

4. Create an Azure Data Lake Gen 2 Storage Account
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Storage
 - d. Pick Storage Account
 - e. Click Create
 - f. Select the right subscription
 - g. Pick the resource group you created above
 - h. Enter a Storage account name
 - i. Select any Location
 - j. Select Standard Performance
 - k. Select Account kind: StorageV2
 - l. Select Locally-redundant Storage (LRS)
 - m. Select the Hot Access tier
 - n. Click Next: Networking
 - o. Click Next: Advanced
 - p. Enable Data Lake Storage Gen2 Hierarchical namespace
 - q. Click Review + Create
 - r. Click Create
 - s. Once the resource is created...
 - t. Click the Containers button
 - u. Click + File system
 - v. Create a crime container

Key Vault

5. Create a Key Vault in the **North Central US** region
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Vault
 - d. Choose Key Vault

- e. Click Create
- f. Select the right subscription
- g. Pick the resource group you created above
- h. Give it a name
- i. Select the **North Central US** Region
- j. Select the Standard pricing tier
- k. Click Create

Application Insights

6. Create an Application Insights resource in the **North Central US** region
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Application Insights
 - d. Click Create
 - e. Select the right subscription
 - f. Pick the resource group you created above
 - g. Give it a name
 - h. Select the **North Central US** Region
 - i. Click Create

SQL Server

7. Create a SQL Server (logical server)
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for SQL Server
 - d. Choose SQL Server (logical server)
 - e. Click Create
 - f. Select the right subscription
 - g. Pick the resource group you created above
 - h. Give it a name
 - i. Select a Location
 - j. Create a Server admin login and password and add them as Secrets to the Key Vault
 - k. Click Next: Networking
 - l. Change the Allow Azure Services to access this server to Yes
 - m. Click Create

Databricks

8. Create a Databricks Service
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Databricks
 - d. Choose Azure Databricks
 - e. Click Create
 - f. Give it a name
 - g. Select the right subscription
 - h. Pick the resource group you created above

- i. Select a Location
- j. Pick the Standard Pricing Tier
- k. Leave the Deploy Databricks in your own VNet at No
- l. Click Create

Container Registry

9. Create a Container Registry
 - a. Open the Resource Group you created above.
 - b. Click the Add button
 - c. Search for Container Registry
 - d. Choose Container Registry
 - e. Click Create
 - f. Give it a name
 - g. Select the right subscription
 - h. Pick the resource group you created above
 - i. Select a Location
 - j. Do Enable the Admin User
 - k. Pick the Basic SKU
 - l. Click Create

Machine Learning

10. Create a Machine Learning service BUT NOT THROUGH THE PORTAL
 - a. Instead, open the Azure Cloud Shell in the browser:



- b. Then, select a Bash prompt. If you've never used the Cloud Shell, follow the prompts to set up a new storage account to use for the Shell.
- c. Afterwards, the Shell will load:

```

Bash  |  [power] [?] [gear] [upload] [download] [code] [share]
Requesting a Cloud Shell. Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

simon@Azure:~$
    
```

- d. Enter:


```
az login
```
- e. Follow the instructions to login.

- f. Enter the following commands:


```
az extension add -n azure-cli-ml
az extension add -n application-insights
```
- g. Now enter this to set the default subscription:


```
az account set -s "<Subscription Name>"
```
- h. Then, enter this to make sure that the Container service is registered:


```
az provider show -n Microsoft.ContainerInstance -o table
```
- i. If the table that displays shows the RegistrationState is Not Registered, then enter:


```
az provider register -n Microsoft.ContainerInstance
```
- j. Now, enter this to get the storage account's id:


```
az storage account show --name <storage-account-name> --query "id" --resource-group <resource-group-name>
```

 Where <storage-account-name> is the Blob Storage account you created earlier.
- k. Copy the ID to a notepad window. It will look like this:


```
"/subscriptions/f2f49d01-fdfd-406e-9d34-4c0e70e9db8c/resourceGroups/Synapse/providers/Microsoft.Storage/storageAccounts/omwtmmlstorage"
```
- l. Enter:


```
az monitor app-insights component show --app <appinsights-account-name> --query "id" --resource-group <resource-group-name>
```

 Where <appinsights-account-name> is the Application Insights account you created earlier.
- m. Copy the ID to the notepad window.
- n. Enter:


```
az keyvault show --name <key-vault-name> --query "id" --resource-group <resource-group-name>
```

 Where <key-vault-name> is the Key Vault you created earlier.
- o. Copy the ID to the notepad window.
- p. Enter:


```
az acr show --name <container-registry-name> --query "id" --resource-group <resource-group-name>
```

 Where <container-registry-name> is the Container Registry you created earlier.
- q. Copy the ID to the notepad window.
- r. Now, in the notepad window, assemble the following statement and then enter it in the console:


```
az ml workspace create -w <workspace-name> -g <resource-group-name> -l northcentralus --keyvault "<key-vault-ID>" --application-insights "<app-insights-ID>" --storage-account "<storage-account-ID>" --container-registry "<container-registry-ID">
```

With (almost)¹ all our resources configured, we can proceed through the Exercises.

¹ We'll be adding a Data Warehouse later on, but everything else is configured.

Exercise 1: Getting Some Data to Analyze

The FBI has some really cool Crime data that is relatively accessible through its Uniform Crime Reporting program: <https://www.fbi.gov/services/cjis/ucr/>

For an example of how the Crime data is used, check out the Crime Data Explorer:

<https://crime-data-explorer.fr.cloud.gov/explorer/state/tennessee/>

The real goldmine is in the Downloads and Documentation page of the Crime Data Explorer:

<https://crime-data-explorer.fr.cloud.gov/downloads-and-docs>

Here, you can download incident-based data by state and year. I downloaded, Tennessee data for 2009 to 2018. However, in 2016, they changed the file formats so I ended up only loading data from 2009 to 2015.

Unzipping one of the files, you find a `sqlite_setup.sql` file and a `sqlite_load.sql` script.

Download and install SQLite, if you don't already have it.

<https://www.sqlite.org/2019/sqlite-dll-win64-x64-3300100.zip>

<https://www.sqlite.org/2019/sqlite-tools-win32-x86-3300100.zip>

<https://github.com/pawelsalawa/sqlitestudio/releases/download/3.2.1/InstallSQLiteStudio-3.2.1.exe>

At a command prompt, enter:

```
sqlite3 your_db_name.db < sqlite_create.sql
```

Then to load in data into the database, run the following in each extracted zipfile of data you have downloaded

```
sqlite3 your_db_name.db < sqlite_load.sql
```

You can create a batch file to load all the unzipped files into one database:

```
cd TN-2010
sqlite3 ..\Crime.db < sqlite_load.sql
cd ..
cd TN-2011
sqlite3 ..\Crime.db < sqlite_load.sql
cd ..
cd TN-2012
sqlite3 ..\Crime.db < sqlite_load.sql
cd ..
cd TN-2013
sqlite3 ..\Crime.db < sqlite_load.sql
cd ..
cd TN-2014
sqlite3 ..\Crime.db < sqlite_load.sql
cd ..
```

Once the data is loaded, use SQLite Studio to create a view to bring all the data together:

NOTE: If you don't want to go through this Exercise, you can simply download the required CSV file from my blog: <https://onemanwenttomow.files.wordpress.com/2020/01/crimeoverview100k.zip>

```

create view CrimeOverview
as
select
  m.data_year as year
  , m.month_num as month
  , ifnull(off_age.age_name, 'Unknown') as off_age_name
  , off_age_num as off_age
  , ifnull(off_ethnicity.ethnicity_name, 'Unknown') as off_ethnicity_name
  , case off.sex_code when '' then 'U' else ifnull(off.sex_code, 'U') end as off_sex
  , ifnull(off_race.race_desc, 'Unknown') as off_race
  , i.incident_date
  , i.incident_hour
  , ot.offense_name
  , ot.crime_against
  , ot.offense_category_name
  , ul.suspect_using_name
  , ifnull(wt.weapon_name, 'Unknown') as weapon_name
  , c.state_abbr as state
  , vt.victim_type_name
  , v.age_num as victim_age
  , ifnull(victim_age.age_name, 'Unknown') as victim_age_name
  , ifnull(victim_race.race_desc, 'Unknown') as victim_race
  , case (v.sex_code) when '' then 'U' else ifnull(v.sex_code, 'U') end as victim_sex
  , ifnull(victim_ethnicity.ethnicity_name, 'Unknown') as victim_ethnicity_name
  , ifnull(r.relationship_name, 'Unknown') as off_vic_relationship_name
  , ifnull(cir.circumstances_name, 'Unknown') as victim_circumstances_name
  , ifnull(injury.injury_name, 'Unknown') as victim_injury_name
  , ifnull(jf.justifiable_force_name, 'Unknown') as victim_justifiable_force_name
  , ifnull(loc.location_name, 'Unknown') as location_type_name
  , ifnull(cat.criminal_act_name, 'Unknown') as criminal_act_type_name
  , ifnull(bias_type.bias_name, 'Unknown') as bias_type_name
from
  nibrs_incident i
  left join nibrs_month m
    on i.nibrs_month_id = m.nibrs_month_id
  left join nibrs_offender off
    on i.incident_id = off.incident_id
  left join nibrs_age off_age
    on off.age_id = off_age.age_id
  left join ref_race off_race
    on off.race_id = off_race.race_id
  left join nibrs_ethnicity off_ethnicity
    on off.ethnicity_id = off_ethnicity.ethnicity_id
  left join nibrs_offense o
    on i.incident_id = o.incident_id
  left join nibrs_offense_type ot
    on o.offense_type_id = ot.offense_type_id
  left join nibrs_suspect_using su
    on o.offense_id = su.offense_id
  left join nibrs_using_list ul
    on su.suspect_using_id = ul.suspect_using_id
  left join nibrs_weapon w
    on o.offense_id = w.offense_id
  left join nibrs_weapon_type wt
    on w.weapon_id = wt.weapon_id
  left join cde_agencies c
    on i.agency_id = c.agency_id
  left join nibrs_victim v

```

```

    on i.incident_id = v.incident_id
left join nibrs_victim_type vt
  on v.victim_type_id = vt.victim_type_id
left join nibrs_age victim_age
  on v.age_id = victim_age.age_id
left join ref_race victim_race
  on v.race_id = victim_race.race_id
left join nibrs_ethnicity victim_ethnicity
  on v.ethnicity_id = victim_ethnicity.ethnicity_id
left join nibrs_victim_offender_rel rel
  on off.offender_id = rel.offender_id
  and v.victim_id = rel.victim_id
left join nibrs_relationship r
  on rel.relationship_id = r.relationship_id
left join nibrs_victim_circumstances victim_cir
  on v.victim_id = victim_cir.victim_id
left join nibrs_circumstances cir
  on victim_cir.circumstances_id = cir.circumstances_id
left join nibrs_victim_injury victim_injury
  on v.victim_id = victim_injury.victim_id
left join nibrs_injury injury
  on victim_injury.injury_id = injury.injury_id
left join nibrs_justifiable_force jf
  on victim_cir.justifiable_force_id = jf.justifiable_force_id
left join nibrs_location_type loc
  on o.location_id = loc.location_id
left join nibrs_criminal_act ca
  on o.offense_id = ca.offense_id
left join nibrs_criminal_act_type cat
  on ca.criminal_act_id = cat.criminal_act_id
left join nibrs_bias_motivation bias
  on o.offense_id = bias.offense_id
left join nibrs_bias_list bias_type
  on bias.bias_id = bias_type.bias_id

```

Then export the results of the view to a standard csv file.

Exercise 2: Loading the Data Lake

Here we will use the Azure Storage Explorer to upload the data file to Blob storage, then we will use Azure Data Factory (ADF) to load the file into the Data Lake. This exercise could be simplified by just uploading the file to the Data Lake using Storage Explorer, but in the interest of learning how, we will use ADF, which is more likely in a real-world example.

Step 1: Upload the file to Blob storage

Since we're simulating the process of loading data, we're going to just use the Azure Portal to upload the csv file created in Exercise 1 to Blob storage in Azure.

First, in the Azure Portal, open the Blob Storage account you created in the Setup section.

1. Click the big Containers button on the Blob Storage overview.
2. Click +Container to add a new container.
3. Give it a name (such as, input)
4. Set it to Private (no anonymous access)

5. Click OK.

Second, open the newly created container:

1. Click the Upload button.
2. Locate the csv file you created or downloaded and unzipped in Exercise 1.
3. Click Ok. Click Upload.

Step 2: Use Data Factory to Transfer the Blob to the Data Lake

Typically, you will use Data Factory as your ETL tool to load data into the Data Lake where the data is staged for further processing with Databricks.

In Data Factory, click Author & Monitor, then select the Author button on the left.

First, let's set up two connections:

1. Select Connections in the bottom left corner.
2. Click +New
3. Select Azure Blob Storage
4. Give your connection a name
5. Select Account Key for Authentication Method
6. Select your Subscription
7. Select your Blob Storage Account
8. Click Test connection
9. Click Create
10. Click +New again
11. Select Azure Data Lake Storage Gen2
12. Give your connection a name
13. Select Account Key for Authentication Method
14. Select your Subscription
15. Select your ADLS Storage Account
16. Click Test connection
17. Click Create

Second, let's create two Datasets:

1. Next to the filter box, click the + and select Dataset.
2. Choose Azure Blob Storage as the data store
3. Click Continue
4. Choose DelimitedText as the format
5. Click Continue
6. Give the Dataset a name
7. Select your Blob Storage linked service
8. Under File path, browse to select the Container name you uploaded the file to and then select the csv file you uploaded
9. Check the First row as header box
10. Click OK
11. Click on the Connection tab

12. Click the Preview data button
13. If that works, go on to the next step
14. Create the second Dataset. Next to the filter box, click the + and select Dataset.
15. Select Azure Data Lake Storage Gen2
16. Click Continue
17. Select Parquet (a common file format for the Data Lake and Databricks)
18. Click Continue
19. Give your Dataset a name
20. Select the Data Lake connection
21. Enter crime for the File System
22. Leave the Directory blank
23. Enter CrimeOverview.parquet as the File
24. Leave Import schema at None
25. Click OK

Third, we'll create the copy pipeline:

1. Next to the filter box, click the + and select Pipeline.
2. Give your Pipeline a name
3. In the center, under Move & transform, drag a Copy data task into the Pipeline
4. Give the Copy task a name
5. Select the Source tab
6. Select the csv Dataset
7. Uncheck the Recursively checkbox
8. Select the Sink tab
9. Select the parquet Dataset
10. Select the Mapping tab
11. Click Import schemas
12. At this point, you could clean up the data types, we're going to leave them all as strings and deal with the data types in Databricks

Finally, click the Publish All button to Publish everything to the Data Factory.

Then trigger the copy pipeline by selecting Add trigger, Trigger now. After it has spun up, you can see the progress by selecting the Monitor button on the left.

Exercise 3: Processing the data in Databricks

Now we have the data as a parquet file in the Data Lake. We need to load the data into some tool, parse, filter and shape it, then load it into a Machine Learning model for analysis.

One tool (among many), that we can do this with is Databricks. It has native support for parquet files and provides access to the Spark Machine Learning libraries.

First, open the Databricks service you created earlier. Click the Launch Workspace button in the middle of the Overview page.

Next, click on the Clusters icon on the left.

1. Click Create Cluster
2. Give your cluster a name
3. Pick Standard as the Cluster Mode
4. Pick None for Pool
5. Pick Runtime: 6.2 (Scala 2.11, Spark 2.4.4)
6. Uncheck the Autoscaling checkbox
7. Change the Terminate timeout to 30 minutes
8. Leave the Worker Type at Standard_DS3_v2
9. Decrease the number of Workers to 2
10. Click Create Cluster at the top
11. Now, select the cluster to open it.
12. Select the Libraries tab
13. Click Install New
14. Pick PyPi as the library source
15. Enter azureml-sdk[databricks] as the Package name
16. Leave Repository blank
17. Click Install

Now, while that's all spinning up, click on the Workspace icon on the left.

1. Right-click on the whitespace under the Workspace heading and pick Create > Notebook.
2. Give the Notebook a name
3. Make sure that Python is selected for the Language
4. Click Create

Databricks notebooks are very similar to Jupyter notebooks. You type code into the box (called a Cell) and run it by pressing Ctrl-Enter or clicking the Play button.

The notebook keeps the context from each Cell for use in other Cells. Using a notebook, you can code a bit, run a bit, code a bit, run a bit, and so on.

Connecting to the Data Lake

The first thing we need to do is bring in the data. For this, we need to connect the notebook to the Data Lake. For that we need an access key²:

1. Open the Data Lake in another tab
2. Select Access keys
3. Copy key1 to the clipboard
4. Switch back to the notebook
5. In the first cell, enter this code:

```
adl_key = "<paste in Data Lake Access Key here">
```

² Access keys are nowhere near as secure as other methods for connecting to a Data Lake. However, most of those require upgrading to Premium services. So, for the sake of simplicity, we'll just be using the key. At the very least, you should follow [these instructions](#) to connect your Databricks environment to an Azure Key Vault and store the Access key there.

```
spark.conf.set("fs.azure.account.key.<insert_data_lake_name>.dfs.core.windows.net", adl_key)
```

6. Press ctrl-enter to run the code

After some time, the notebook will have spun up the cluster, submitted the code to the cluster and executed it. You should get some feedback that the Command took 0.10 seconds or so.

Now we have a connection between the notebook/cluster and the Data Lake.

Loading the Parquet File

In the next cell, enter and execute the following code to display the files from the crime File System in the Data Lake:

```
src = "abfss://crime@<insert_data_lake_name>.dfs.core.windows.net/"
dbutils.fs.ls(src)
```

abfss is a special protocol that tells the cluster to look in the Data Lake for a file.

Now that we can see the file we transferred into the Data Lake, we can load it into a Spark View (which is a type of temp table). Enter and execute the following code:

```
spark.read.parquet(src +
"crime.parquet").createOrReplaceTempView("CrimeOverview")
```

And to see the data in the Spark View, we can use Spark SQL. Enter and execute the following code:

```
df = spark.sql("""select cast(year as int) year, cast(month as int) month,
    cast(off_age as int) off_age,
    off_sex, off_race, cast(incident_date as date) incident_date,
    offense_name,
    offense_category_name, suspect_using_name, weapon_name, state,
    cast(victim_age as int) victim_age, victim_sex, victim_race,
    off_vic_relationship_name, location_type_name
    from CrimeOverview
    where crime_against = 'Person'
    and victim_type_name = 'Individual'
    and off_age_name = 'Age in Years'
    and victim_age_name = 'Age in Years'
    and off_sex <> 'U' and off_race <> 'Unknown'
    and victim_sex <> 'U'
    and victim_race <> 'Unknown' """)
df = df.dropna() # drops any rows with null values in them
df.head()
```

Finally, let's cut down the resultset to just the columns we're going to need for the Machine Learning exercise. Enter and execute this code:

```
colsToKeep = ['year', 'month', 'victim_age', 'victim_sex', 'offense_category_name',
              'location_type_name', 'off_vic_relationship_name']
df2 = df.select(*colsToKeep)
df2.head()
```

And that's it. You've loaded the data into a Spark dataset ready for running through the Machine Learning model, which is the next exercise.

Exercise 4: Creating the Machine Learning Model

If you're not familiar with Machine Learning, then this exercise will likely be Greek to you. However, I have tried to add comments to the code to explain what is going on.

Basically, we would like the Machine to predict what the relationship between the offender and the victim is most likely to be, given a specific month (year and month), a specific victim (age and sex), an offense category and a location.

For example, if a 24-year-old male is a victim of theft at a store, who is the most likely suspect? His brother, his friend, a stranger, who?

First, some setup. Enter and execute this code to configure and login to the Azure ML Workspace:

```
from datetime import datetime
from azureml.core import Workspace
from azureml.core import Experiment

region = "northcentralus"
subscriptionid = "7810278c-6fd2-48d8-b77e-135508bea8b6"
resourceGroupName = "Synapse2020"
workspaceName = "synapse2020-ml"
experimentName = "experiment-" + datetime.now().strftime("%j%H%M")
ws = Workspace.get(name=workspaceName, subscription_id=subscriptionid,
                  resource_group=resourceGroupName)
exp = Experiment(workspace=ws, name=experimentName)

#NOTE:: This requires a manual login in a Standard Databricks Account
```

When you execute the code for the first time, it will pause and wait for you to do an interactive authentication. There will be a prompt at the bottom of the code block. You will need to open another browser window to the specified address and enter the code you are given. This is to authenticate to the Azure ML instance.

Now, enter and execute this code to build and test the Machine Learning Model:

```
from pyspark.ml import Pipeline, PipelineModel
```

```

from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import IndexToString, StringIndexer, VectorAssembler

run = exp.start_logging()
labelIdx_sex = StringIndexer(
    inputCol="victim_sex", outputCol="victim_sex_idx").fit(df2)
labelIdx_offense_category_name = StringIndexer(inputCol="offense_category_name",
    outputCol="offense_category_name_idx").fit(df2)
labelIdx_off_vic_relationship_name = StringIndexer(inputCol="off_vic_relationship_name",
    outputCol="off_vic_relationship_name_idx").fit(df2)
labelIdx_location_type_name = StringIndexer(inputCol="location_type_name",
    outputCol="location_type_name_idx").fit(df2)
# assemble the input vectors for use in the model as a column called 'features'
assembler = VectorAssembler(inputCols=['year', 'month', 'victim_age',
    'victim_sex_idx', 'offense_category_name_idx', 'location_type_name_idx'],
    outputCol="features")
# use a Random Forest Classifier on the features to create the predictions
rfr = RandomForestClassifier(featuresCol="features",
    labelCol="off_vic_relationship_name_idx",
    predictionCol="prediction", maxBins = 45)
# turn the encoded predictions back into normal text
inverter = IndexToString(inputCol="prediction",
    outputCol="predicted_relationship",
    labels=labelIdx_off_vic_relationship_name.labels)
# create a pipeline that runs all the steps above
pipeline = Pipeline(stages=[labelIdx_sex, labelIdx_offense_category_name,
    labelIdx_off_vic_relationship_name, labelIdx_location_type_name, assembler,
    rfr, inverter])
# split up the dataset into 70% training and 30% testing
(trainingData, testData) = df2.randomSplit([0.7, 0.3])
# fit the model with the training data
model = pipeline.fit(trainingData)
# use the model with the test data to make predictions
predictions = model.transform(testData)
predictions.select("prediction", "predicted_relationship", "features").show(5)
# calculate the Accuracy
evaluator = MulticlassClassificationEvaluator(
    labelCol='off_vic_relationship_name_idx',
    predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
run.log('Accuracy', accuracy)
run.log('Error', 1.0 - accuracy)
run.complete()
print(f"Accuracy: {accuracy}")

```

```
print("Test Error = %g" % (1.0 - accuracy))
```

If you're using the same dataset as me, your Accuracy value should be around 0.25. This represents a lousy fit as we have a 75% error rate. So our model is not very good at predicting the offender-victim relationship. I will leave it to you to tweak the model and see if you can get better predictions.

Exercise 5: Detour Ahead! Create a Custom Docker Base Image

It turns out there is a known (and not fixed) bug when deploying an Azure Databricks Machine Learning model to an Azure ACI Container (Docker image) Webservice.

The exact error you will see in if you try to deploy your model to a container now is:

```
Exception: Java gateway process exited before sending the driver its port number
```

This turned out to be a very resilient bug and I spent far too much time banging my head against my monitor to figure out a way to deal with it. Much googling found several articles, including [this one on StackOverflow](#) that talk about the problem without providing any solutions that pertain to this particular deployment scenario.

Eventually, I decided to see if I could install Java in the Webservice image that I was deploying to. It turns out that I could³. Here's how.

Install Docker Desktop on Your Local Machine

The instructions for installing Docker Desktop on your local machine are pretty straight forward:

1. Go to <https://docs.docker.com/docker-for-windows/install/>
2. Click the big "Download from Docker Hub" button
3. Create an account, sign your life away, etc.
4. Download the installer
5. Run the installer
6. Reboot

Clone and Modify a New Docker Image

Now that Docker Desktop is installed, you can create Docker Images. We need to create one that we can use as the base image for our deployed webservice.

Launch a command line and then enter and execute:

```
C:\>docker
```

If you don't get an error, then all is right with the world and we can continue. We're going to base our image on the standard Microsoft image for machine learning webservices. Enter

³ See here for more info: <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-custom-docker-image#create-a-custom-base-image>

```
C:\>docker run -it mcr.microsoft.com/azureml/o16n-sample-user-base/ubuntu-
miniconda
```

Now you should have an image and it's loaded and you're logged in as the admin and you see a weird prompt, like this:

```
root@111222333444:/#
```

You are inside the image now, so let's run some commands to update the image, install Java, update miniconda, and install azureml. Enter and execute the following commands at the root@ prompt:

```
root@111222333444:/# apt-get update && apt-get upgrade
root@111222333444:/# apt-get install software-properties-common
root@111222333444:/# apt-get install vim curl git
root@111222333444:/# apt-get install default-jdk
root@111222333444:/# conda update conda
root@111222333444:/# conda install pyspark
root@111222333444:/# pip install azureml
root@111222333444:/# pip install azureml-core
root@111222333444:/# pip install azureml-defaults
root@111222333444:/# pip install inference-schema
root@111222333444:/# exit
```

Now you should be back at the C:\> prompt.

Next, we need to commit the container we just created, ran and updated as a Docker Image file.

```
C:\> docker commit 111222333444 ubuntu-miniconda-java
```

Now we can upload that new image to the Azure Container Registry we created in the beginning.

```
C:\> docker login <name of container registry>.azurecr.io
Username: <name of container registry>
Password: <Found on Access Keys tab of Container Registry>

C:\> docker tag ubuntu-miniconda-java <name of container
registry>.azurecr.io/ubuntu-miniconda-java

C:\> docker push <name of container registry>.azurecr.io/ubuntu-miniconda-
java
```

And that's it. We have an image to use in the deployment exercise.

To Edit the Docker Image

If, for some reason, you need to edit the image, go back to the command prompt and enter these commands:

```
C:\> docker create -t -i ubuntu-miniconda-java bash
555666777888dhsskapfhqsfouipqyfewyrfewasdakcvsadh

C:\> docker start -a -i 555666777888

root@555666777888:/# DO STUFF
root@555666777888:/# exit

C:\> docker commit 555666777888 ubuntu-miniconda-java
C:\> docker tag ubuntu-miniconda-java <name of container
registry>.azurecr.io/ubuntu-miniconda-java
C:\> docker push <name of container registry>.azurecr.io/ubuntu-miniconda-java
```

Exercise 6: Configure the Model for Deployment

Now that we've created and run a machine learning model, we need to register the model, create an entry point script, and create a configuration file for the Environment. Then we can deploy the model as an ACI Container (Docker image) in the next exercise.

First, we register the Model. To register it we need to save it, copy the saved files to a specific folder, and then register it/them. Enter and execute the following three code blocks:

```
model_name = 'crimemodel'
model.write().overwrite().save('dbfs:/model/' + model_name)
```

```
import os

model_dbfs = 'dbfs:/model/' + model_name
model_local = "file:" + os.getcwd() + "/" + model_name
dbutils.fs.cp(model_dbfs, model_local, True)
dbutils.fs.ls(model_local)
```

```
from azureml.core.model import Model

registeredModel = Model.register(
    model_path=model_name, model_name=model_name, workspace=ws)
```

Now we need to create the Entry Point script. This is a Python file that the Docker Image will use to interface with the Machine Learning model we just registered. To do this, we can enter and execute the following code:

```
py = """ import os
import json
```

```

import pandas as pd
from pyspark.ml import PipelineModel
from pyspark.sql.session import SparkSession

from inference_schema.schema_decorators import input_schema, output_schema
from inference_schema.parameter_types.pandas_parameter_type import
PandasParameterType

def init():
    global model
    # Replace filename if needed.
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), '*****' + model_name +
    "****")
    # Deserialize the model file back into a spark model.
    spark_session = SparkSession.builder.getOrCreate()
    model = PipelineModel.load(model_path)

input_sample = pd.DataFrame([
    {"year": 2015,
    "month": 2,
    "victim_age": 24,
    "victim_sex": "M",
    "offense_category_name": "Burglary",
    "location_type_name": "Home"
    },{
    "year": 2016,
    "month": 3,
    "victim_age": 45,
    "victim_sex": "F",
    "offense_category_name": "Assault",
    "location_type_name": "Work"
    }])

output_sample = pd.DataFrame([
    {"year": 2015,
    "month": 2,
    "victim_age": 24,
    "victim_sex": "M",
    "offense_category_name": "Burglary",
    "location_type_name": "Home",
    "predicted_relationship": "Friend"
    },{
    "year": 2016,

```

```

    "month": 3,
    "victim_age": 45,
    "victim_sex": "F",
    "offense_category_name": "Assault",
    "location_type_name": "Work",
    "predicted_relationship": "Spouse"
  })
})

@input_schema('data', PandasParameterType(input_sample))
@output_schema(PandasParameterType(output_sample))
def run(data):
    try:
        spark_session = SparkSession.builder.getOrCreate()
        df = pd.DataFrame(data)
        spark_input = spark_session.createDataFrame(df)
        result = model.transform(spark_input)
        predictions = result.toPandas()
        predictions = predictions[['year', 'month', 'victim_age', 'victim_sex',
                                   'offense_category_name', 'location_type_name',
                                   'predicted_relationship']]
        return predictions.to_dict(orient='records')
    except Exception as e:
        error = str(e)
        return error
"""
dbutils.fs.put("dbfs:/models/test/score.py", py, overwrite=True)

```

Lastly, we need to create a configuration file for the Environment we will create in the next exercise. To do this, we can enter and execute the following code:

```

cy = """channels:
- conda-forge
dependencies:
- python=3.6
- pyspark=2.4.4
- pandas
- pip:
  - azureml-core==1.0.81.1
  - azureml-defaults==1.0.81
  - inference-schema
  - inference-schema[pandas-support]
  - pynacl==1.2.1
name: env"""
dbutils.fs.put("dbfs:/models/test/conda.yaml", cy, overwrite=True)

```

That's it for this exercise. We now have all the bits we need to be able to deploy the model as a webservice.

Exercise 7: Build and Deploy the Docker image

With the model registered and all the bits ready, we can create the Environment we need to deploy the model from. To do this, we can enter and execute the following code:

```
from azureml.core.environment import Environment

# Create the environment
myenv = Environment.from_conda_specification(name = "test-
env", file_path = "/dbfs/models/test/conda.yaml")

# Enable Docker and reference the custom image from Exercise 5
myenv.docker.enabled = True
myenv.docker.base_image = "<Container Registry Name>.azurecr.io/ubuntu-miniconda-
java"

#this comes from the Container Registry Access keys
myenv.docker.base_image_registry.address = "<Container Registry Name>.azurecr.io"
myenv.docker.base_image_registry.username = "<Container Registry Name>"
myenv.docker.base_image_registry.password = "<password from Container Registry>"

# This will install the inference specific apt packages.
myenv.inferencing_stack_version = "latest"

myenv.register(workspace=ws)
```

With the Environment registered, we are ready to deploy the model as a Webservice, like so:

```
from azureml.core.webservice import AciWebservice, Webservice
from azureml.exceptions import WebserviceException
from azureml.core.model import InferenceConfig

service_name = 'franks' # yes, as in Frankenstein, with a nod to Larry Correia
try:
    Webservice(ws, service_name).delete()
except WebserviceException:
    pass

dc = AciWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)
ic = InferenceConfig(entry_script="/dbfs/models/test/score.py", environment=myenv
)
```

```
webservice = Model.deploy(ws, service_name, [registeredModel], ic, dc, overwrite=True)
```

The deployment will take 3-10 minutes (or more) to run, you can monitor it with this code:

```
webservice.wait_for_deployment()
print(webservice.get_logs())
```

Another option for seeing what is going on is to open the Container instance (once it shows up in your resource group, which could take a few minutes):

Resources			
Synapse2020			
			Refresh
synapse2020containers	Container registry	North Central US	
synapse2020-ml	Machine Learning	North Central US	
synapse2020adl	Storage account	North Central US	
synapse-adf	Data factory (V2)	North Central US	
synapse2020blobs	Storage account	North Central US	
synapse2020-appinsights	Application Insights	North Central US	
franks	Container instances	West US	
synapse2020-kv	Key vault	North Central US	
synapse2020-sqlserver	SQL server	North Central US	

Inside the container, click on Containers. Select the second one (franks) and click on the Logs tab. Eventually, this will update and you will see the deployment log.

Now, with a healthy dose of good fortune, the Webservice will deploy without error. If not, I leave it to you to debug it (sorry). Double-check the steps in this document and hopefully, the environment won't have changed so much that these instructions no longer apply.

With the model deployed as a webservice, we can proceed to the next exercise, which is to create a client and test the webservice.

Exercise 8: Testing the Webservice

To test the webservice, we're going to need some data to pass in to the model for analysis. Here, we will use Spark SQL again to see whodunnit for a variety of crimes and locations when the victim is a 45 year-old male, in December of 2016. Like so:

```
dfTest = spark.sql("""select distinct 2016 year, 12 month, 45 victim_age,
                        'M' victim_sex, offense_category_name, location_type_name
                        from CrimeOverview
                        where crime_against = 'Person' and victim_type_name = 'Individual'""")
```

```

and off_age_name = 'Age in Years'
and victim_age_name = 'Age in Years' and
off_sex <> 'U' and off_race <> 'Unknown' and victim_sex <> 'U'
and victim_race <> 'Unknown'
""")
dfTest = dfTest.dropna()
data = dfTest.toPandas()
data.head()

```

Which yields a pandas dataframe that looks like this:

	year	month	victim_age	victim_sex	offense_category_name	location_type_name
0	2016	12	45	M	Sex Offenses	Specialty Store
1	2016	12	45	M	Assault Offenses	Convenience Store
2	2016	12	45	M	Kidnapping/Abduction	Convenience Store
3	2016	12	45	M	Homicide Offenses	Parking Lot/Garage
4	2016	12	45	M	Sex Offenses	Bar/Nightclub

We can then send that data into the webservice and, if the stars align, get the predicted relationship of the offender with this code:

```

test_sample = json.dumps({'data': data.to_dict(orient='records')})

scoring_uri = webservice.scoring_uri
response = requests.post(
    url=scoring_uri, data=test_sample,
    headers={"Content-type": "application/json"})
response_json = json.loads(response.text)
resultdf = pd.DataFrame(response_json)
resultdf = resultdf[['year', 'month', 'victim_age', 'victim_sex',
                    'offense_category_name', 'location_type_name',
                    'predicted_relationship']]
resultdf.head()

```

Which *should* result in a pandas dataframe like this:

	year	month	victim_age	victim_sex	offense_category_name	location_type_name	predicted_relationship
0	2016	12	45	M	Sex Offenses	Specialty Store	Victim Was Acquaintance
1	2016	12	45	M	Assault Offenses	Convenience Store	Victim Was Acquaintance
2	2016	12	45	M	Kidnapping/Abduction	Convenience Store	Victim Was Acquaintance
3	2016	12	45	M	Homicide Offenses	Parking Lot/Garage	Victim Was Acquaintance
4	2016	12	45	M	Sex Offenses	Bar/Nightclub	Victim Was Stranger

Whew. That was a long journey. The end is in sight now though. Next up, push a bunch of data through to Power BI.

A couple of things to know

To figure out what the input parameters and output parameters of the webservice should look like, you can view the “swagger” schema document at this url:

```
print(webservice.swagger_uri)
```

The url will look something like this:

<http://11696b61-9a58-4529-bc6d-7dab4d6a3b4b.westus.azurecontainer.io/swagger.json>

And it yields document that shows this:

```

▼ definitions:
  ▼ ServiceInput:
    type: "object"
    ▼ properties:
      ▼ data:
        type: "array"
        ▼ items:
          type: "object"
          ▼ properties:
            ▼ year:
              type: "integer"
              format: "int64"
            ▼ month:
              type: "integer"
              format: "int64"
            ▼ victim_age:
              type: "integer"
              format: "int64"
            ▼ victim_sex:
              type: "string"
            ▼ offense_category_name:
              type: "string"
            ▼ location_type_name:
              type: "string"
          ▼ example:
            ▼ data:
              ▼ 0:
                year: 2015
                month: 2
                victim_age: 24
                victim_sex: "M"
                offense_category_name: "Burglary"
                location_type_name: "Home"
              ▼ 1:
                year: 2016
                month: 3
                victim_age: 45
                victim_sex: "F"
                offense_category_name: "Assault"
                location_type_name: "Work"

```

If you forgot to note the swagger_uri or the scoring_uri for your webservice, you can recover them with this code:

```
from azureml.core import Webservice
from azureml.core import Workspace

region = "northcentralus"
subscriptionid = "7810278c-6fd2-48d8-b77e-135508bea8b6"
resourceGroupName = "Synapse2020"
workspaceName = "synapse2020-ml"
ws = Workspace.get(name=workspaceName, subscription_id=subscriptionid,
                  resource_group=resourceGroupName)

services = Webservice.list(ws)
print(services[0].scoring_uri)
print(services[0].swagger_uri)
```

Exercise 9: Loading the Data Warehouse

By using a combination of Data Factory and Databricks, we can transfer the historical crime data to the data warehouse and we can create a future facing dataset that uses our ML model to predict outcomes going forward. In this case, we have 2009 to 2015 data and we will predict 2016 data.

Create the Data Warehouse

1. Create a new Azure Synapse Analytics (Data Warehouse) .
 - a. Open the Resource Group you created earlier
 - b. Click the Add button
 - c. Click on Databases on the left
 - d. Select the Azure Synapse Analytics (formerly SQL DW) option
 - e. Select the right subscription
 - f. Pick the resource group you created above
 - g. Enter a name for your Data Warehouse
 - h. Select the SQL Server we created earlier
 - i. Click Select Performance Level
 - j. Scale down to DW100c
 - k. Click Apply
 - l. Click Review + create
 - m. Click Create

NOTE: The DW costs \$1.50 PER HOUR. Do not leave it running if you are not using it and DELETE it when you are done with these exercises.

Create the Databricks ETL Script

In databricks, spin up the cluster and create a new Notebook. In the new Notebook, first we need to connect to the Data Lake and pull in the raw data:

```
adl_key = "<data lake key>"
spark.conf.set("fs.azure.account.key.<data lake
name>.dfs.core.windows.net", adl_key)

src = "abfss://crime@<data lake name>.dfs.core.windows.net/"

spark.read.parquet(src + "crime.parquet").createOrReplaceTempView("CrimeOverview"
)
```

Now we need to fix the data types in the data and write the results back to the Data Lake:

```
df = spark.sql("""select cast(year as int) year, cast(month as int) month,
    cast(off_age as int) off_age,
    off_sex, off_race, cast(incident_date as date) incident_date,
    offense_name,
    offense_category_name, suspect_using_name, weapon_name, state,
    cast(victim_age as int) victim_age, victim_sex, victim_race,
    off_vic_relationship_name, location_type_name,
    victim_type_name, off_age_name, victim_age_name, crime_against
from CrimeOverview""")
df = df.dropna() # drops any rows with null values in them

df.write.parquet(src + "crime_fixed.parquet", mode='overwrite')
```

Next, we're going to construct the dataset for the future. We'll start by getting the distinct values for offense category and location:

```
ocn = spark.sql("""select distinct
    offense_category_name
from CrimeOverview
where crime_against = 'Person'
and victim_type_name = 'Individual'
and off_age_name = 'Age in Years'
and victim_age_name = 'Age in Years'
and off_sex <> 'U' and off_race <> 'Unknown'
and victim_sex <> 'U'
and victim_race <> 'Unknown'""")
ltn = spark.sql("""select distinct
    location_type_name
from CrimeOverview
where crime_against = 'Person'
```

```

        and victim_type_name = 'Individual'
        and off_age_name = 'Age in Years'
        and victim_age_name = 'Age in Years'
        and off_sex <> 'U' and off_race <> 'Unknown'
        and victim_sex <> 'U'
        and victim_race <> 'Unknown'""")
locn = ocn.toPandas()["offense_category_name"].to_list()
lltn = ltn.toPandas()["location_type_name"].to_list()

```

Next, we can construct a pandas dataframe with all the possible options for 2016, like so:

```

import pandas as pd
# year month offense_category_name victim_age victim_sex location_type_name
lst = []
pdfs = []
year = 2016
for month in range(1, 13):
    for ocname in locn:
        for ltname in lltn:
            for gen in ['M', 'F']:
                for age in range(10, 100):
                    lst.append({"year":year, "month":month, "victim_age":age,
                                "victim_sex":gen, "offense_category_name":ocname, "location_
type_name": ltname})
                    pdf = pd.DataFrame(lst)
                    lst = []
                    pdf = pdf[["year", "month", "victim_age", "victim_sex", "offense_category_n
ame", "location_type_name"]]
                    pdfs.append(pdf)

```

The pdfs list now contains 2700 dataframes of 180 rows that we will feed through ML webservice with this code:

```

import requests
import json
import pandas as pd

def fixDate(row):
    return str(row["year"]) + '-' + ('00' + str(row["month"]))[-2:] + '-01'

scoring_uri = "http://11696b61-9a58-4529-bc6d-7dab4d6a3b4b.westus.azurecontainer.io/score"
bigdf = pd.DataFrame()
for p in pdfs:

```

```

test_sample = json.dumps({'data': p.to_dict(orient='records')})
response = requests.post(
    url=scoring_uri, data=test_sample,
    headers={"Content-type": "application/json"})
response_json = json.loads(response.text)
resultdf = pd.DataFrame(response_json)
resultdf["year_month"] = resultdf.apply(lambda row: fixDate(row), axis=1)
resultdf = resultdf[['year', 'month', 'year_month', 'victim_age',
    'victim_sex', 'offense_category_name', 'location_type_name',
    'predicted_relationship']]
bigdf = bigdf.append(resultdf)

```

The bigdf dataframe now contains all 486,000 predictions. Let's save it back to the data lake:

```

df = spark.createDataFrame(bigdf)
df.write.parquet(src + "crime_future.parquet", mode='overwrite')

```

Now then, when databricks writes out a parquet file, it actually makes a folder with a bunch of part*.parquet files and a few other files that are detritus. To clean this up, run this code:

```

files = dbutils.fs.ls(src + 'crime_fixed.parquet')
for file in files:
    if file.name[0] == '_':
        dbutils.fs.rm(src + 'crime_fixed.parquet/' + file.name)
files = dbutils.fs.ls(src + 'crime_future.parquet')
for file in files:
    if file.name[0] == '_':
        dbutils.fs.rm(src + 'crime_future.parquet/' + file.name)

```

Connect to the DW and Create the Target Tables

Using Azure Data Studio or SSMS, connect to the Azure DW using the SQL Server login you wrote down in the Key Vault earlier.

NOTE: You will have to go to the SQL Server resource in the Azure Portal and edit the Firewall settings to allow your local machine to connect to the SQL Server at all.

Once connected, create two tables:

```

CREATE TABLE [crime].[overview_fixed]
(
    [year] [int] NULL,
    [month] [int] NULL,
    [off_age] [int] NULL,
    [off_sex] [nvarchar](1) NULL,
    [off_race] [nvarchar](250) NULL,
    [incident_date] [datetime2](7) NULL,
    [offense_name] [nvarchar](250) NULL,
    [offense_category_name] [nvarchar](250) NULL,

```

```

[suspect_using_name] [nvarchar](250) NULL,
[weapon_name] [nvarchar](250) NULL,
[state] [nvarchar](2) NULL,
[victim_age] [int] NULL,
[victim_sex] [nvarchar](1) NULL,
[victim_race] [nvarchar](250) NULL,
[off_vic_relationship_name] [nvarchar](250) NULL,
[location_type_name] [nvarchar](250) NULL,
[victim_type_name] [nvarchar](250) NULL,
[off_age_name] [nvarchar](250) NULL,
[victim_age_name] [nvarchar](250) NULL,
[crime_against] [nvarchar](250) NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
GO

CREATE TABLE [crime].[overview_future]
(
    [year] [int] NULL,
    [month] [int] NULL,
    [year_month] [datetime2](7) NULL,
    [offense_category_name] [nvarchar](250) NULL,
    [victim_age] [int] NULL,
    [victim_sex] [nvarchar](1) NULL,
    [location_type_name] [nvarchar](250) NULL,
    [predicted_relationship] [nvarchar](250) NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
GO

```

Back to Data Factory to Load the DW

In Data Factory, create:

- two Azure Data Lake Datasets to the Fixed and Future parquet folders (not files)
- an Azure Synapse Analytics Connection/Linked Service
- two Azure Synapse Analytics Datasets to the overview_fixed and overview_future tables above
- a pipeline with two Copy data activities that copy data from the fixed parquet folder to the overview_fixed table, and from the future parquet folder to the overview_future table, like so:



Note: For the copy activity, under Source, you will need to check Recursively and enter a wildcard file name of *.parquet, and then under Sink, you will need to choose Bulk insert, and enter a Pre-copy script of: TRUNCATE TABLE crime.overview_fixed (or crime.overview_future)

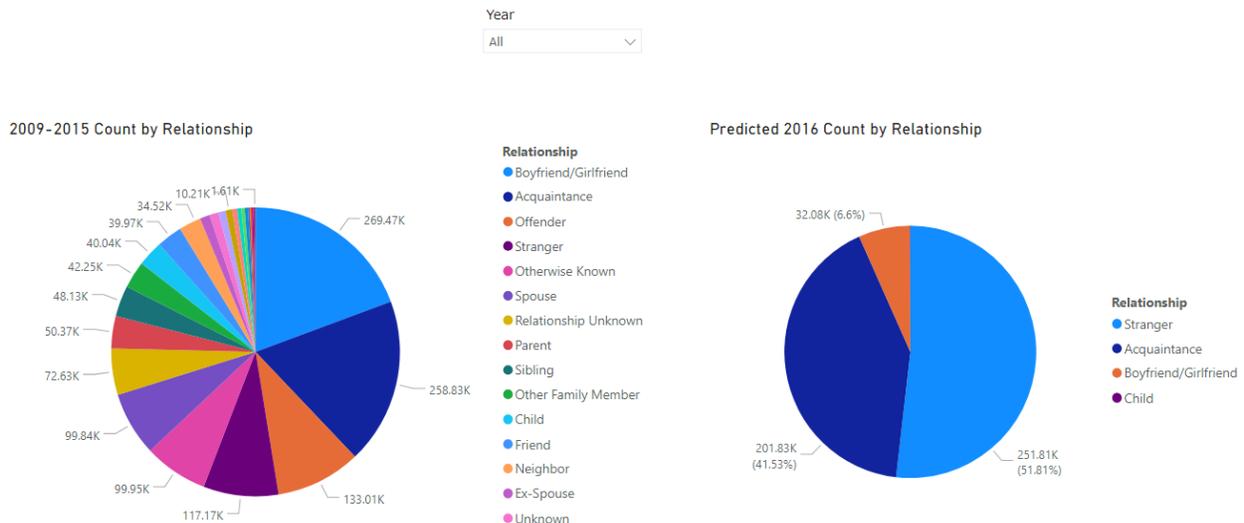
- a pipeline that runs everything with an Execute Pipeline activity to load the CSV from Blob storage to the raw crime.parquet file in the Data Lake, a Databricks Notebook activity that runs our ETL script, and a second Execute Pipeline activity to run the Copy pipeline you just created for the last bullet, like so:



Since we ran the ETL script already, we can just run the new Copy Pipeline that loads the data from the data lake to the data warehouse.

Exercise 10: Creating a Power BI Report

Last but not least, we can create a Power BI report that shows the data we've loaded and predicted.



Remember, our ML model is not very good, so the results don't look great. But this was always about getting the bits to work together, not necessarily to make a good ML model. Maybe I can work on that for next time...