

**MMAI 844**

**Agile Management in AI**

*Group Project – AI in PaaS*

**Creating a Recommender System and Web API**

**using Azure ML Studio and Python**

**Team Osgoode**

|  |  |
| --- | --- |
| **Student Name** | **Student Number** |
| Sneha Abraham | 20197020 |
| Alexander Banh | 06349542 |
| Yao Chen | 20189695 |
| Mahesh Kumar | 20193608 |
| Jerry (He) Liu | 20190907 |
| Darryl Lobo | 20192859 |
| Blair Nicolle | 20194169 |
| Nayef Abou Tayoun | 20141011 |

**Osgoode's Github URL:**

<https://github.com/Osgoode-Queens/AzureML-Recommender-Files>

# Introduction

In this document we cover the creation of a Recommender system in Microsoft Azure ML Studio complete with a web API and webpage. Azure is both an easy-to-use and feature-rich platform.

We've broken down the steps into three (3) sections so that there are logical stopping points to test things out, to allow for experimentation of parameters, and to discuss important concepts:

**Section I:** Creating and Training the recommender model in Azure ML Studio

**Section II:** Creating and Testing the Web API in Azure ML Studio

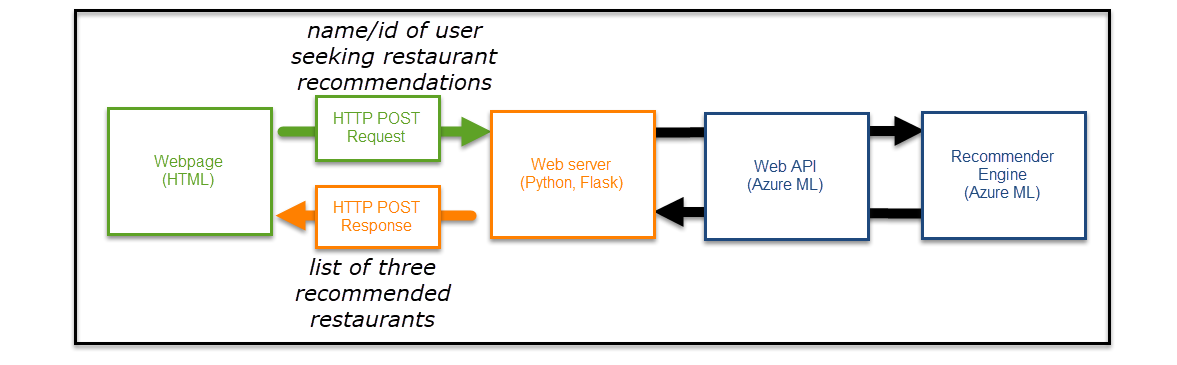
**Section III:** Referencing the Recommender API from a live webpage via Python

In Section I, we'll start with a dataset with people's restaurant ratings data which you can optionally replace with data that is more germane to your business needs (eg movie ratings, product ratings).

Then, in Section II we create an API and unit test it: we will see that when we specify a user id, it will return the top three restaurant recommendations personalized for that particular user.

Finally, in Section III we call the API from a web front end so that you can see the end-to-end result as if it were deployed on the web. We will use the Python library "*Flask*" to act as a web server.

**Summary system illustration:**



Supplemental files and this document are available and cloneable at our public Github page:

<https://github.com/Osgoode-Queens/AzureML-Recommender-Files>

## Expected Results

After you have completed this document, you will have built a simple website which calls a web API that returns restaurant recommendations based on a model that you will train. It will look like this:

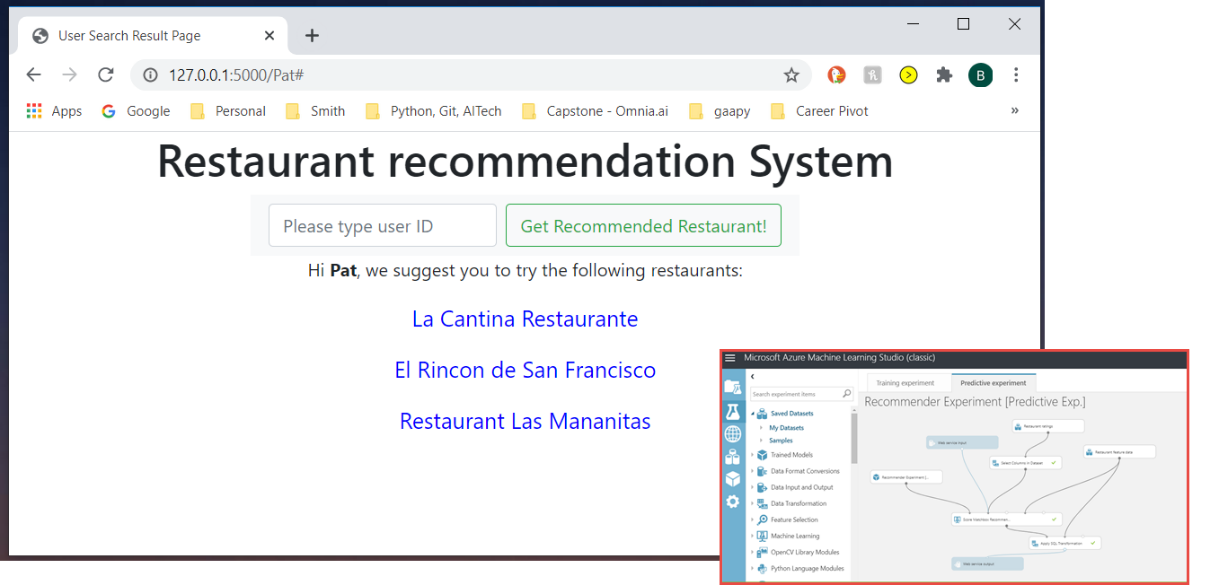


Table of Contents

[Introduction 2](#_Toc44939377)

[Expected Results 3](#_Toc44939378)

[Section I: Creating the system, training the model, and browse results 4](#_Toc44939379)

[Before you Start 4](#_Toc44939380)

[Concepts 5](#_Toc44939381)

[Step by Step 5](#_Toc44939382)

[Section II: Create Web API 20](#_Toc44939383)

[Before you Start 20](#_Toc44939384)

[Concepts 20](#_Toc44939385)

[Step-by-Step 20](#_Toc44939386)

[Section III: Reference Web API 30](#_Toc44939387)

[Concepts 30](#_Toc44939388)

[Step-by-Step 30](#_Toc44939389)

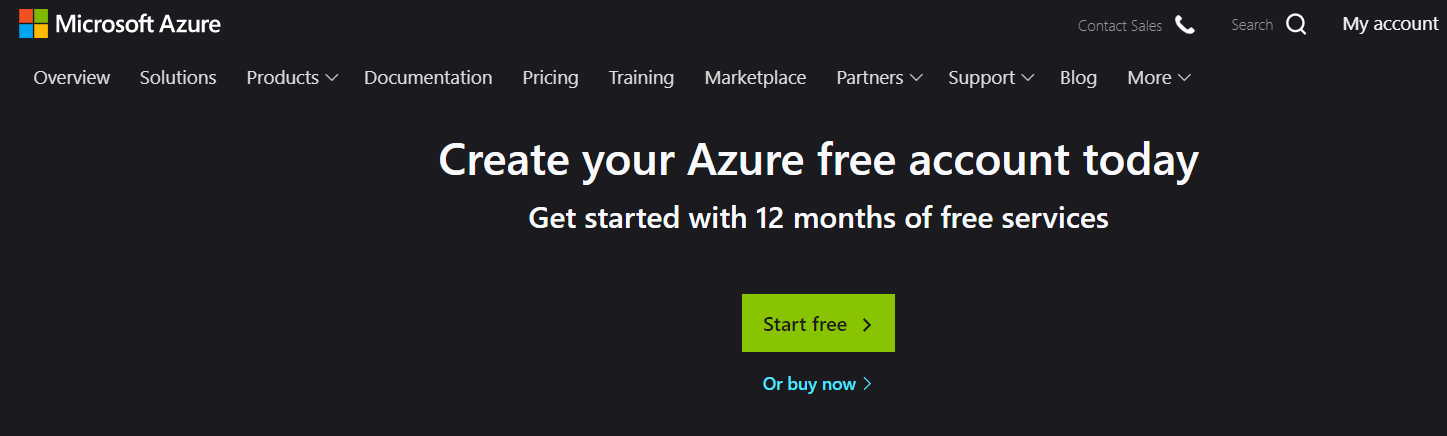
# Section I: Creating the system, training the model, and browse results

## Before you Start

1. Login to your Microsoft Azure account: most of registered an account during MMAI-863.

*...Just in case* you hadn't registered for an Azure account in MMAI-863, the basic steps are below. **Do steps (ii) through (v) only if you don't already have an ML Studio workspace setup. Otherwise, please proceed directly to "Concepts" and "Step-by-Step" sections.**

1. Register for a free Azure account, here: <https://azure.microsoft.com/en-us/free/>



1. We will use the classic **Azure ML Studio**, and you can get there quickest by clicking: <https://studio.azureml.net/> .
2. With Azure ML Studio, create a workspace, within which will reside your machine learning 'Projects' and 'Experiments'.
3. Go into Projects. At this point, you are now ready to start the step-by-step instructions.

## Concepts

There are two common usages of recommender systems:

**Item-by-Item** – Recommendations based on correlation between patterns in the use of items. E.g. Amazon: *"people who bought this item also bought these other items." Or, Netflix: "people who liked this movie also watched these movies."*

**Personalized –** Using collaborative filtering to recommend items based on a particular person's past choices and similarity with choices made by similar people. Amazon: *"Given that you're about to buy <that> item, we think you may also wish to buy <this other> item."*

Our recommender system in this Section could easily be tweaked to perform both types of usages. We will focus on the latter type, for simplicity of mission.

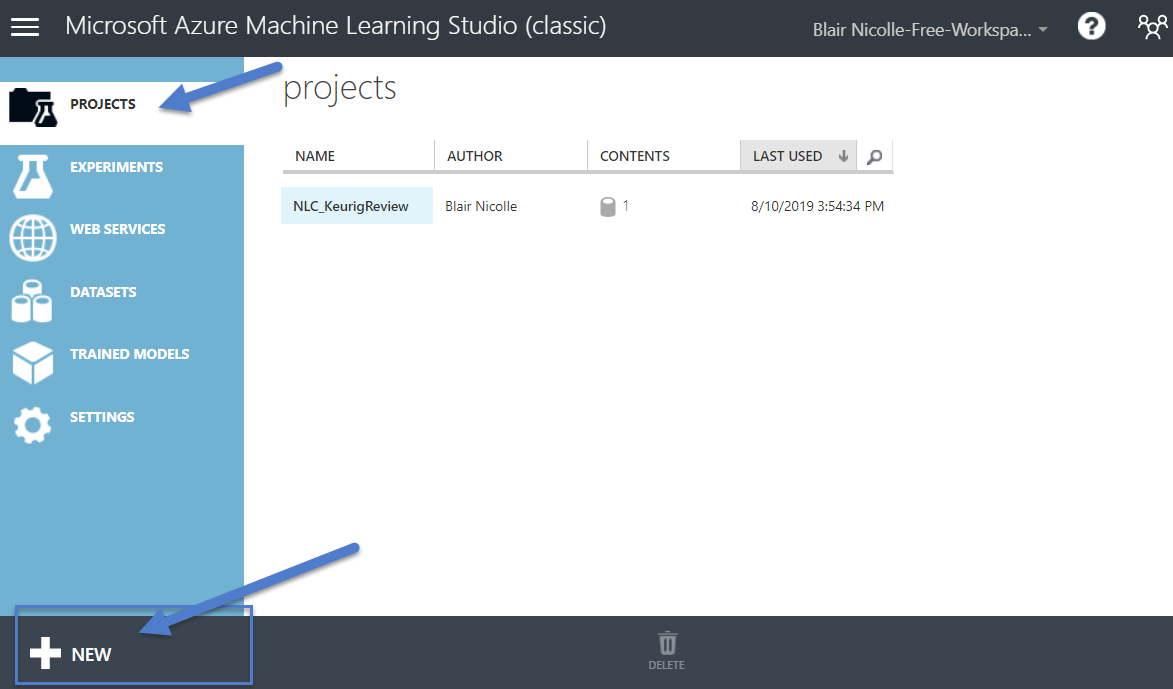
**Note:** Azure's ML Studio UI has been changing and rebranding over time. There is now the so-called "Classic" "Microsoft Azure Machine Learning *Studio*" and the newer "Azure Machine Learning Service." We will use the classic UI, also known as Azure ML Studio.

Azure ML *Studio's* URL is available directly via this link: <https://studio.azureml.net/>

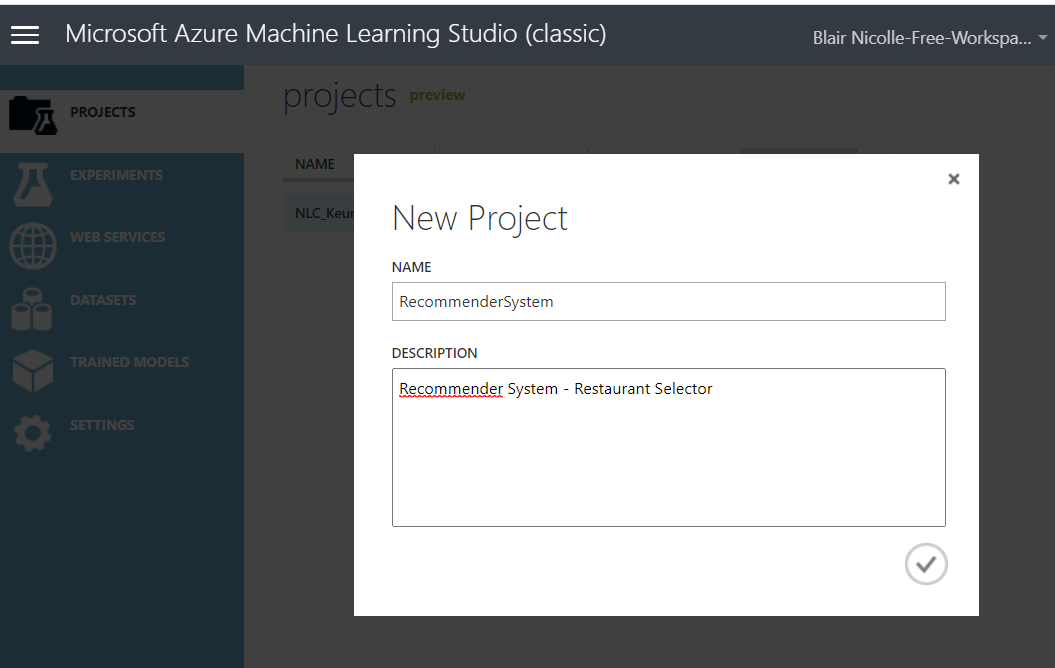
Azure's new ML *Service*'s URL: <https://azure.microsoft.com/en-us/services/machine-learning/>

## Step by Step

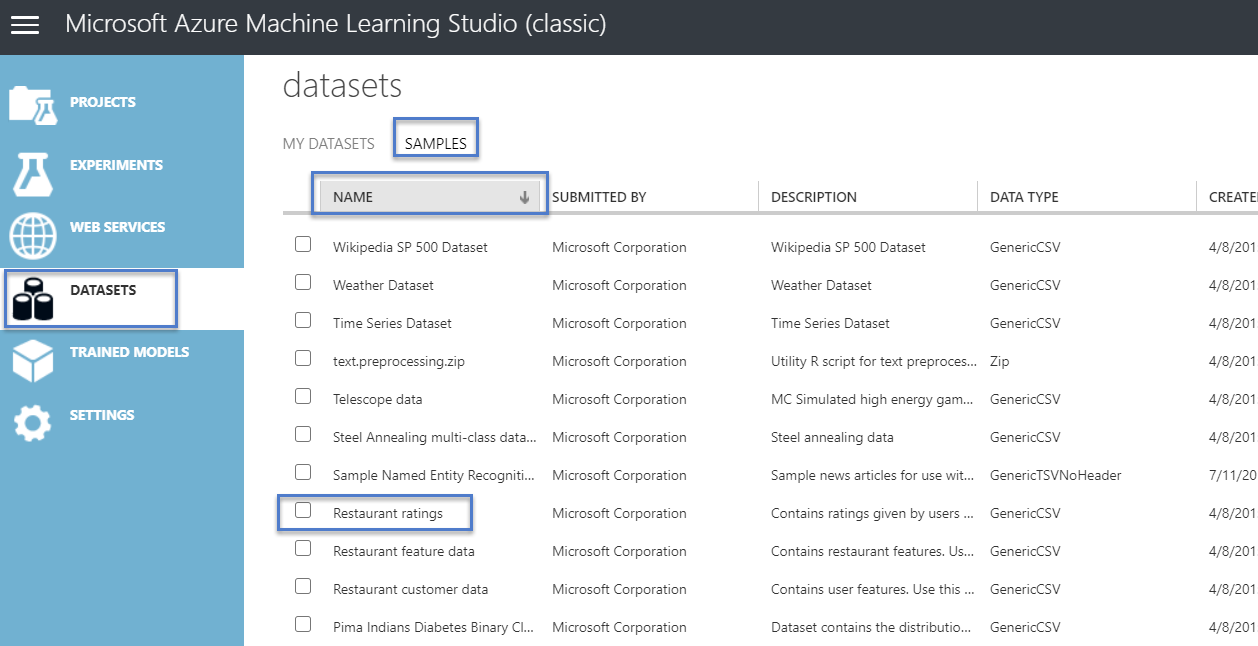
1. **Within Azure ML Studio**, click *Project*s from the left menubar and hit the large "**+**" icon in the lower left of screen to **create a new project.**

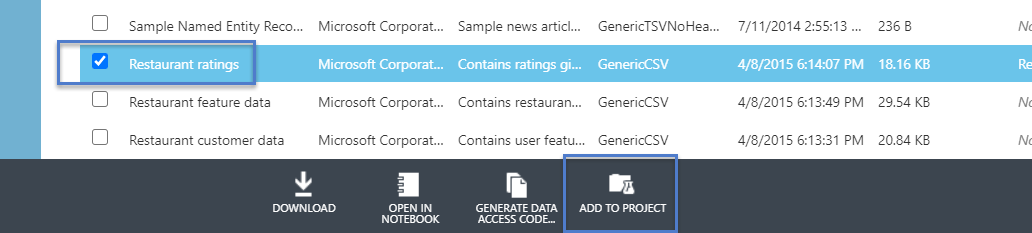


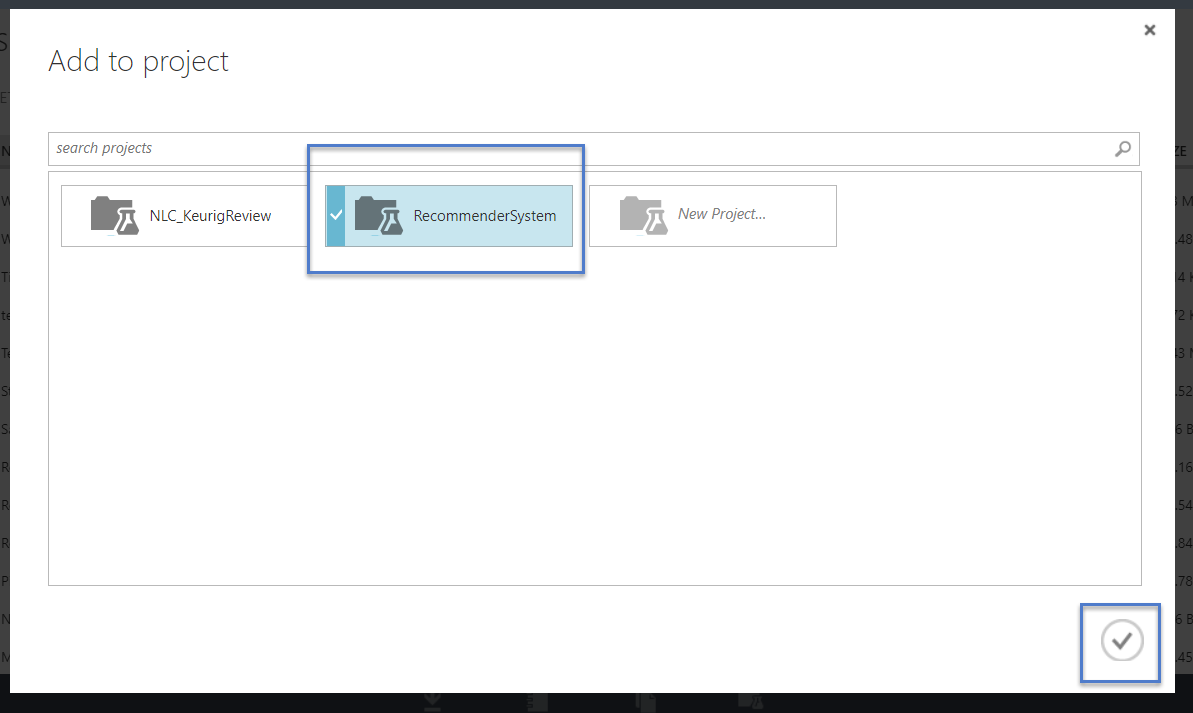
1. Name the new project such as "RecommenderSystem" (spaces are allowed in the name, if you prefer to name it "Recommender System" or "Recommender Project.")



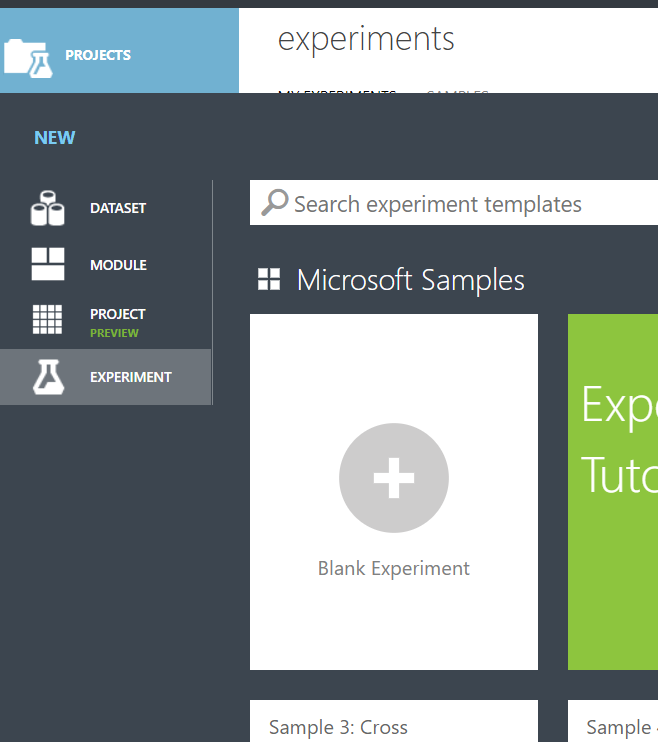
1. Select "Datasets" and select "Restaurant Ratings" from the "Samples" dataset collection. Next, add it to your project by clicking "add to project" and add it to the "RecommenderSystem" project.

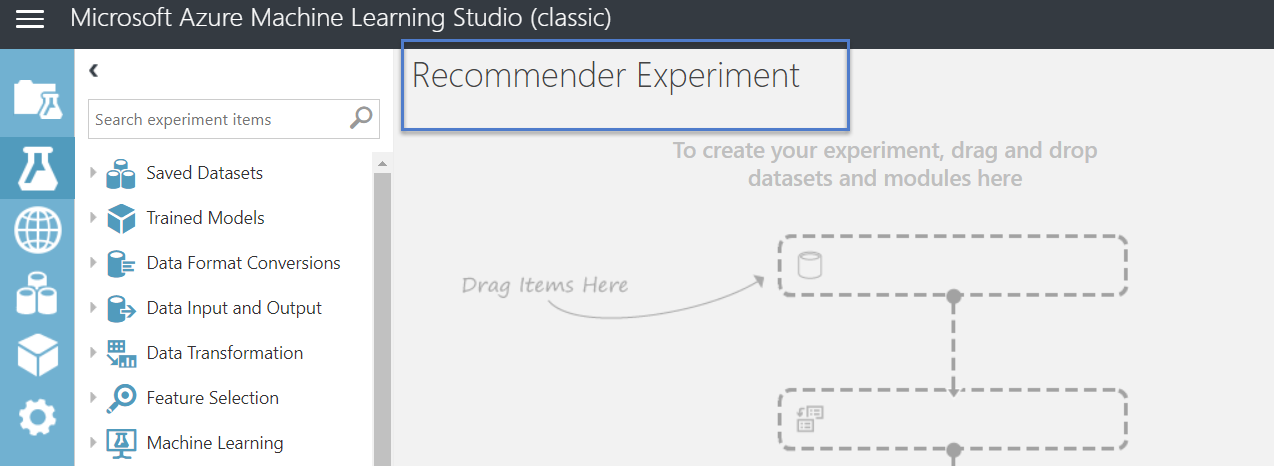




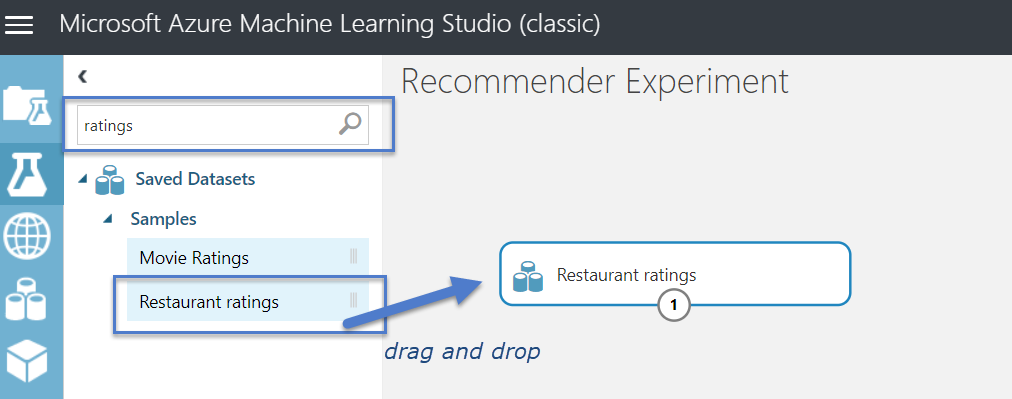


1. Create an Experiment by clicking "Experiment" on side menu, then "+" in lower left hand corner to instantiate it. Use "blank experiment" as a template and name it "Recommender Experiment."

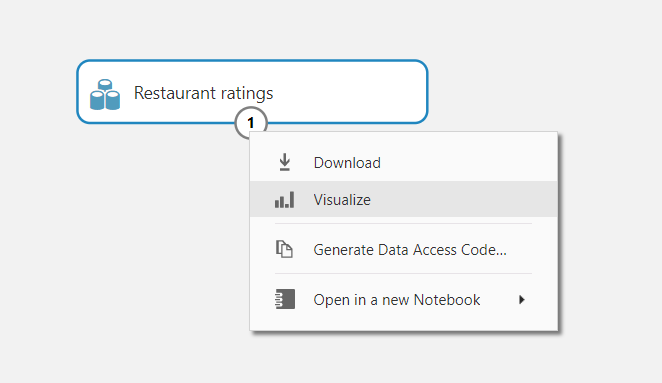


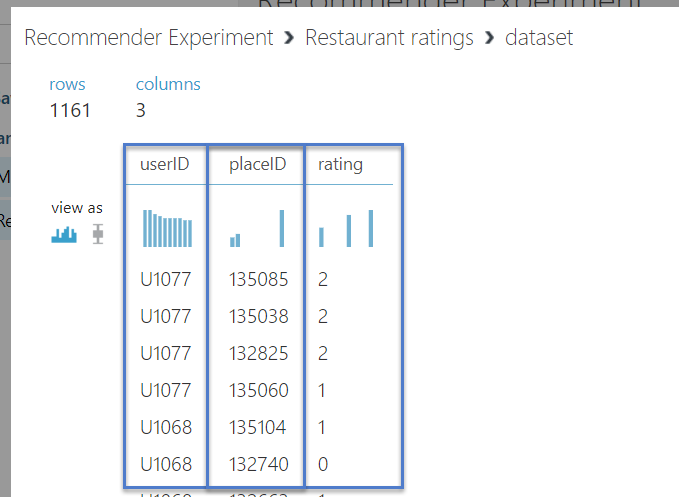


1. In the search box, type "ratings". It will find any object named ratings and you will find the "Restaurant ratings" dataset. Drag this into your experiment.



1. You can browse the dataset by clicking on the circled number 1 and selecting "Visualize." Each row in the dataset represents a different user id's rating (0, 1, or 2) at restaurants they have been to and rated. More granularity in ratings (e.g. 0-10) would require more training data and compute time. The restaurants are just uniquely represented as **placeID**s and the users are just represented as **userID**s.

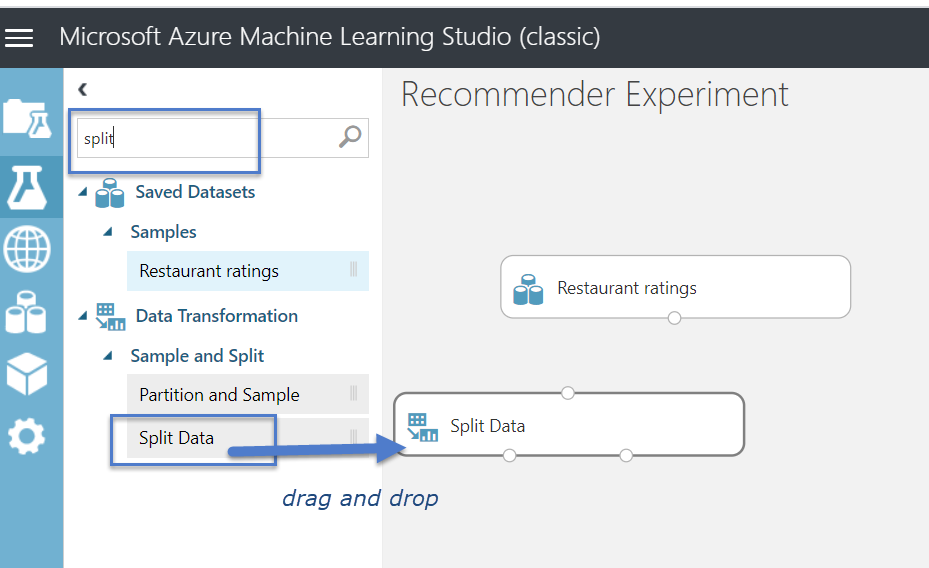


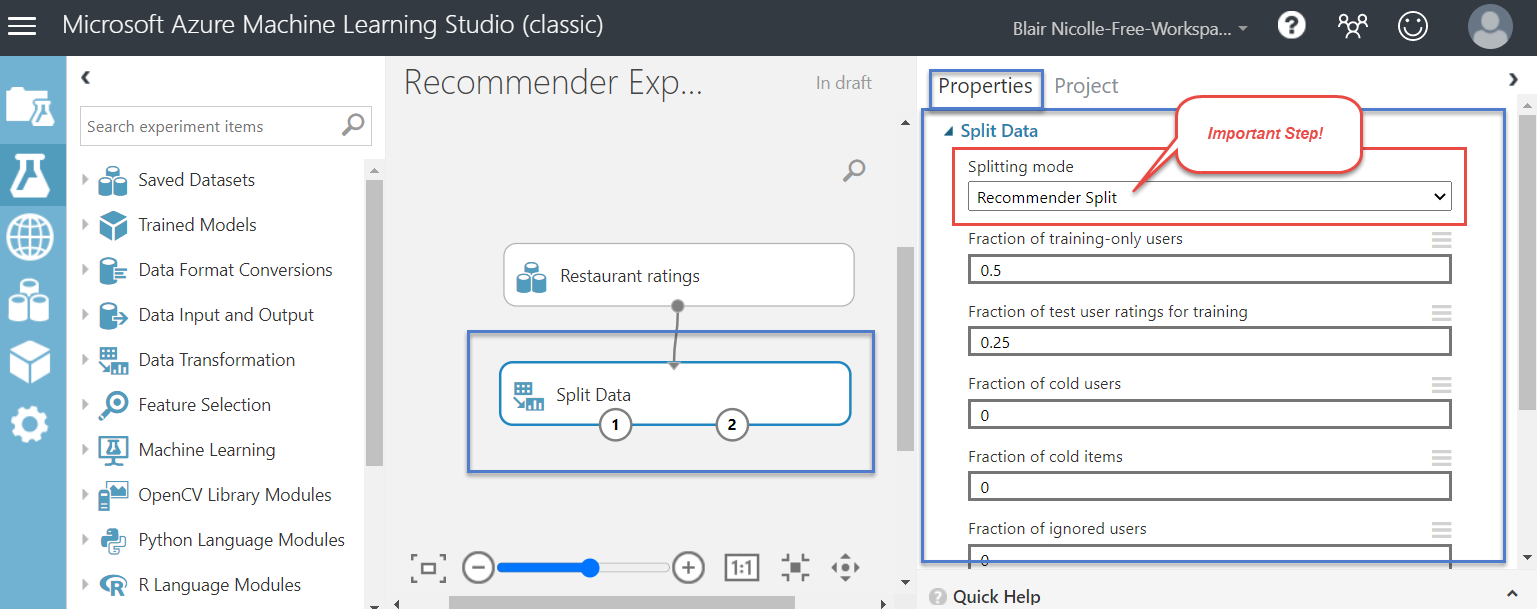


At this point, if you want to choose a different dataset that is more relevant to your purposes (e.g. movie recommendations, book recommendations, product recommendations, etc) then you can specify your dataset. Just use same format – same column layout – as the restaurant ratings dataset.

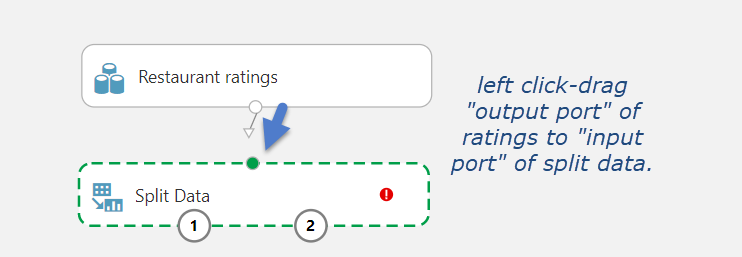
***Advanced:*** If you have a dataset in mind that you want to use, you can use one that references items other than restaurants such as movies or products. Otherwise, you can revisit the datasets later and swap in a more relevant one after you get the basics of ML Studio down cold.

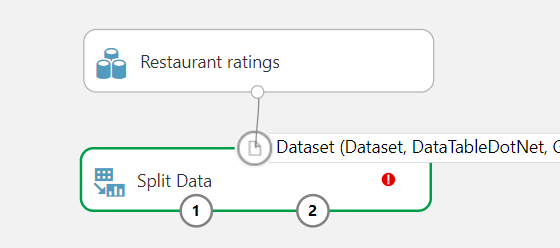
1. Drag in the "Split Data" transformation module. This will allow you to split the dataset into train-test sets. Once the module is in your experiment, you can left click it and view its properties. Here, you can change the "Splitting Mode" property to *"Recommender Split."* You can keep the default split of 75%-**25%** split, change it, and review the other properties available*.*

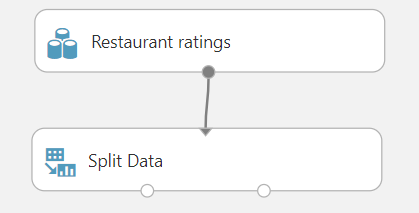




1. Now that you have two **modules** floating around you can connect them to construct the first part of your ML pipeline. Simply connect the 'restaurant ratings' module's **output port** to the 'split data' data transformation module's **input port.** 
   * **Important: Note that the Split Data has two output ports: the left output port is the 'training set' and the right output port represents the 'test set'.**



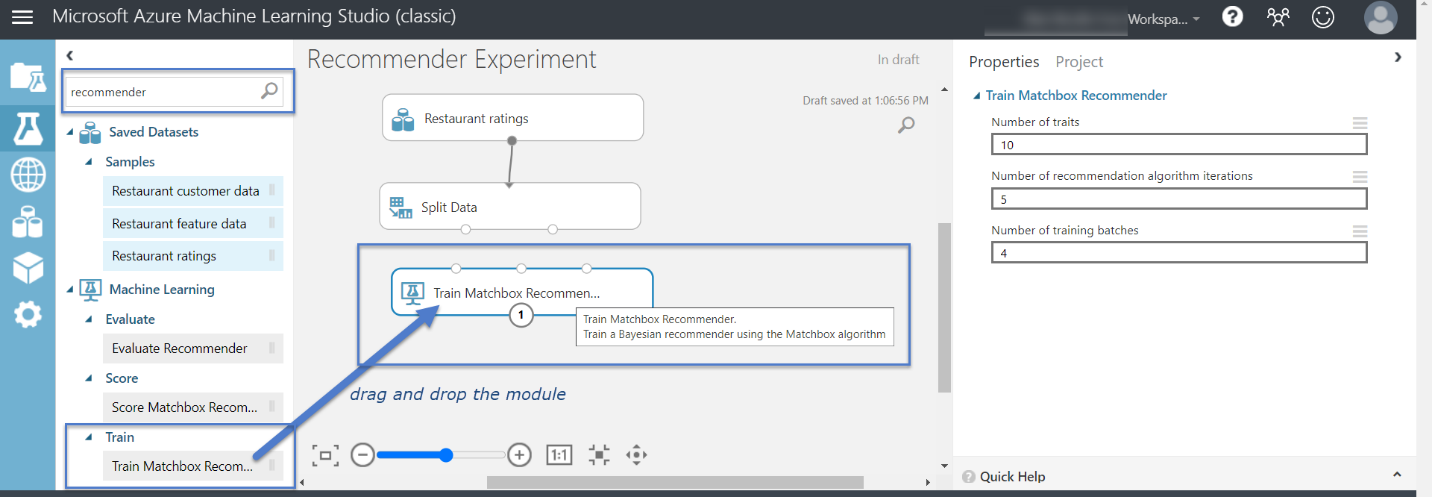




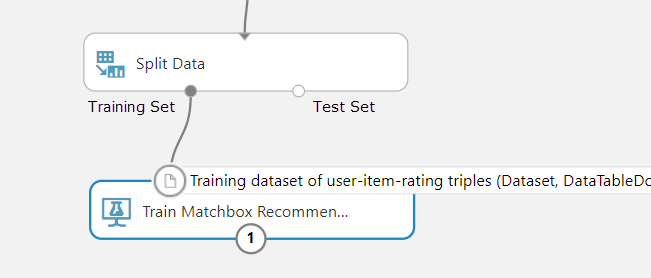
1. Now that you have an (admittedly, very basic) pipeline, you can now click **Save,** which is located on the icon bar at the bottom of the screen. Azure will auto-save occassionally but it's a good habit to keep in mind as you proceed further and when working with pipelines.



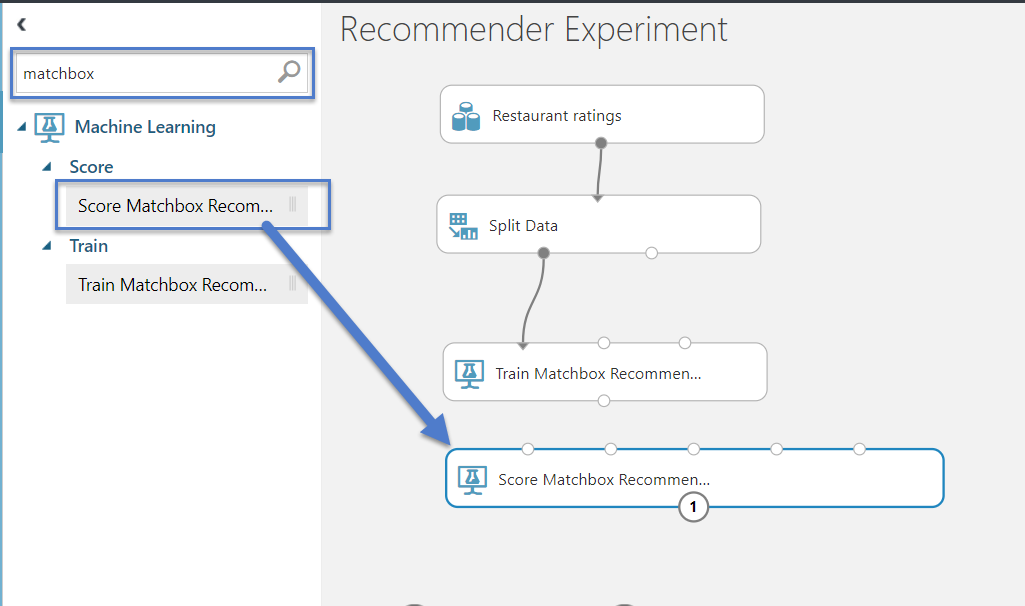
1. Next, we will **train the model.** Search for the keyword "Recommender" or "Matchbox" and drag the "Train Matchbox Recommender". 
   * *Note: Matchbox is the name of a particular algorithm: it is a Bayesian Recommender algorithm.*
   * Once you drag and drop the module into your experiment, you will see the parameters of the Matchbox Recommender's algorithm on the right hand side of the window. We will keep the default values.



1. Connect (drag) the Training Set output port from "Split Data" module to the Left input box on "Train Matchbox Recommender" module.
   * Note: The Train Matchbox Recommender module can handle various scenarios and to improve accuracy. Each scenario has its own input port. The leftmost input port – the one which we are using – is for recommendations based on *user-item-rating* triples.

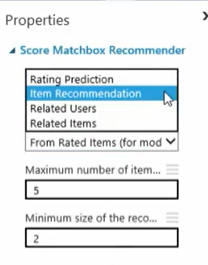
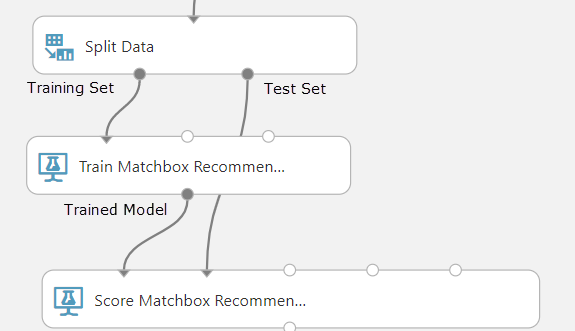


1. We now add the "Score Matchbox Recommender" and wire it in with the Trained model against the Test Data.

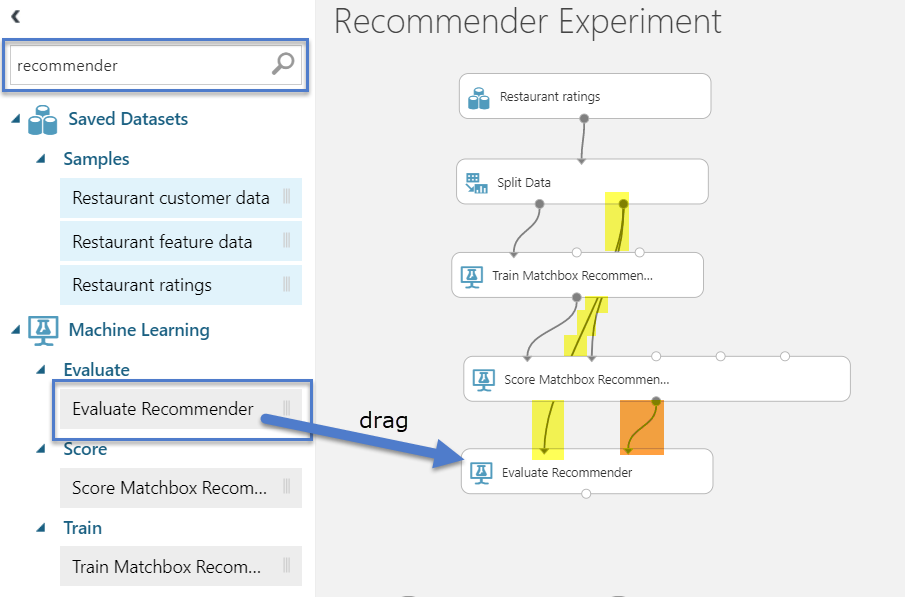


Select the Score Matchbox Recommender module to show the properties panel on top right and select the type of recommendation we wish to do: in this case, "Item Recommendation", which is user-specific item recommendations.

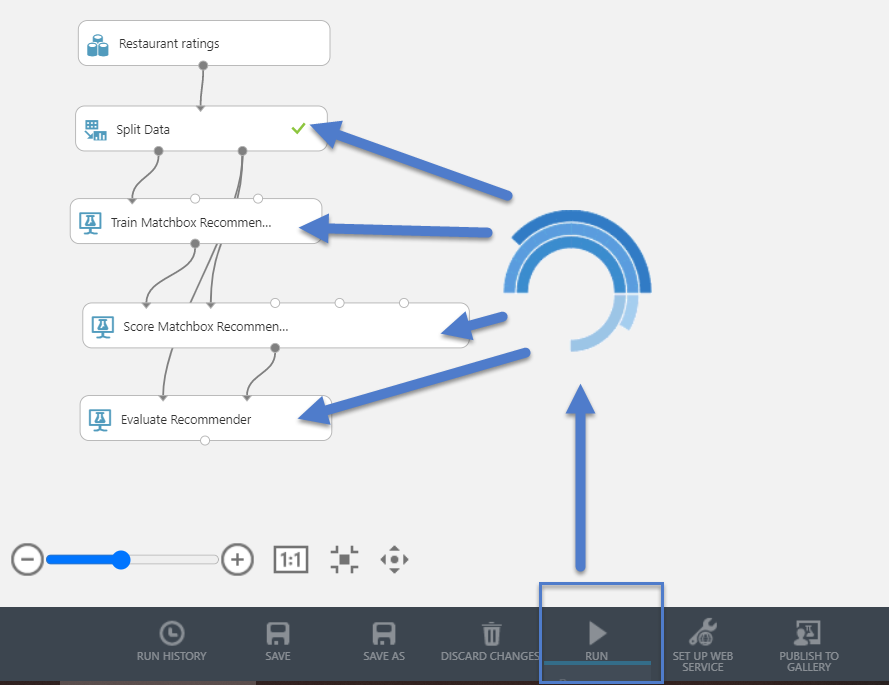
Also, complete the wiring of the Score Matchbox Recommender module as below:

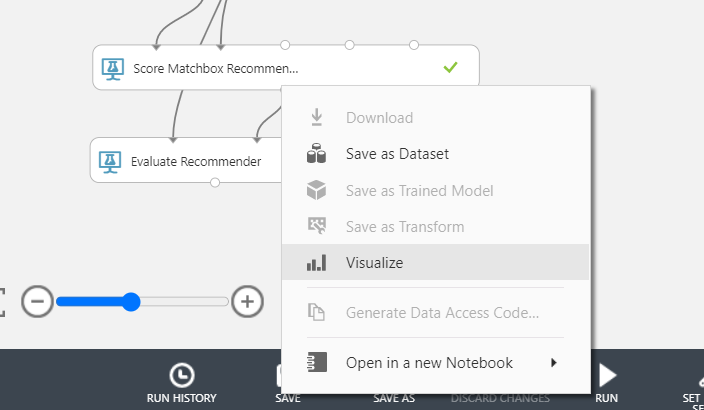
1. Add "Evaluate Recommender" module, wiring the inputs from the Test Set (in yellow highlight) and Scored Test/Validation Set (in orange highlight).



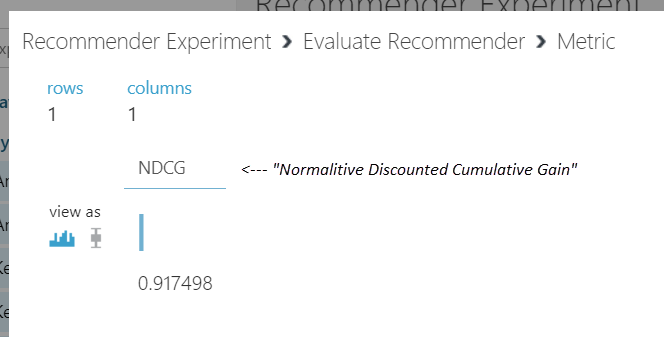
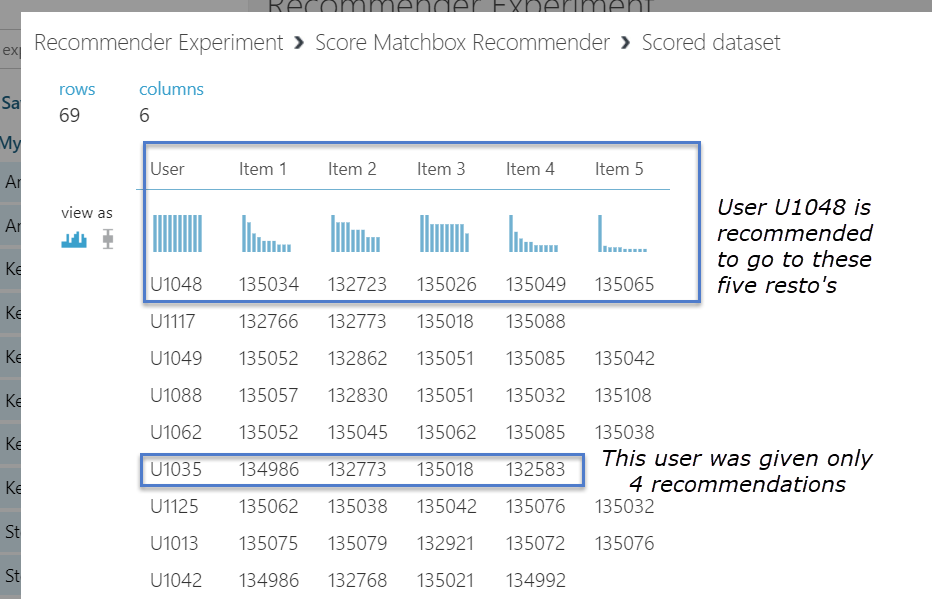
1. **Run the Experiment.**  To remind you, our experiment here will provide restaurant recommendations based on the user's previous ratings; the user's in the default dataset are represnted by unique ids (eg U1234) but we will put names to these identification numbers, shortly. By running the experiment we can see some initial results.
   * **Click Run.**
   * As steps complete, a green checkmark will show.



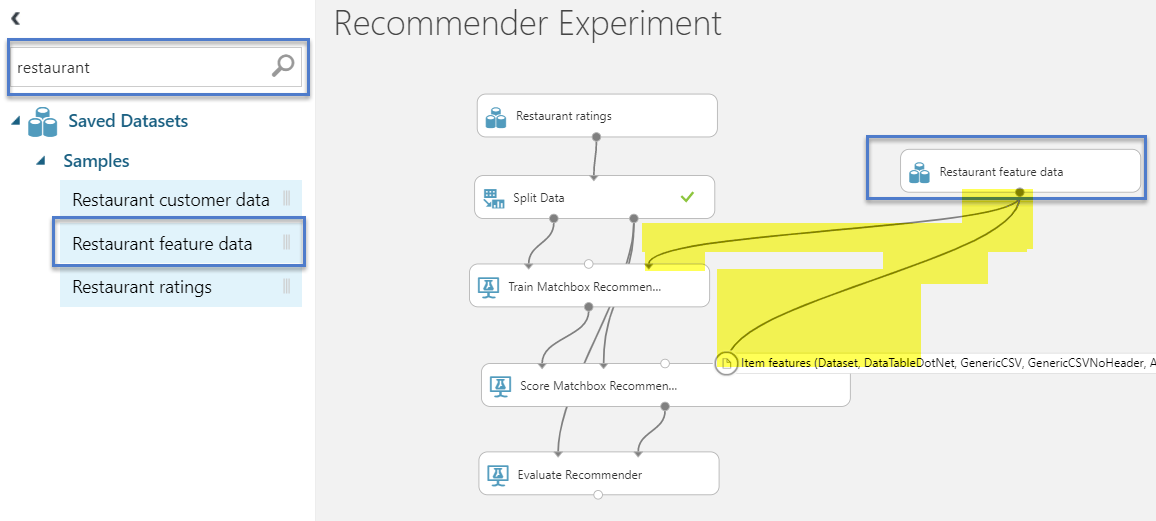
1. View Results:
   * Right click right output port of Score Matchbox Recommender | Visualize shows the scored restaurant recommendations by user id.
   * Right click output port of Evaluate Recommender | Visualize shows a score on a scale from 0 to 1 about how good the recommendations match user preferences (i.e., 0.917).



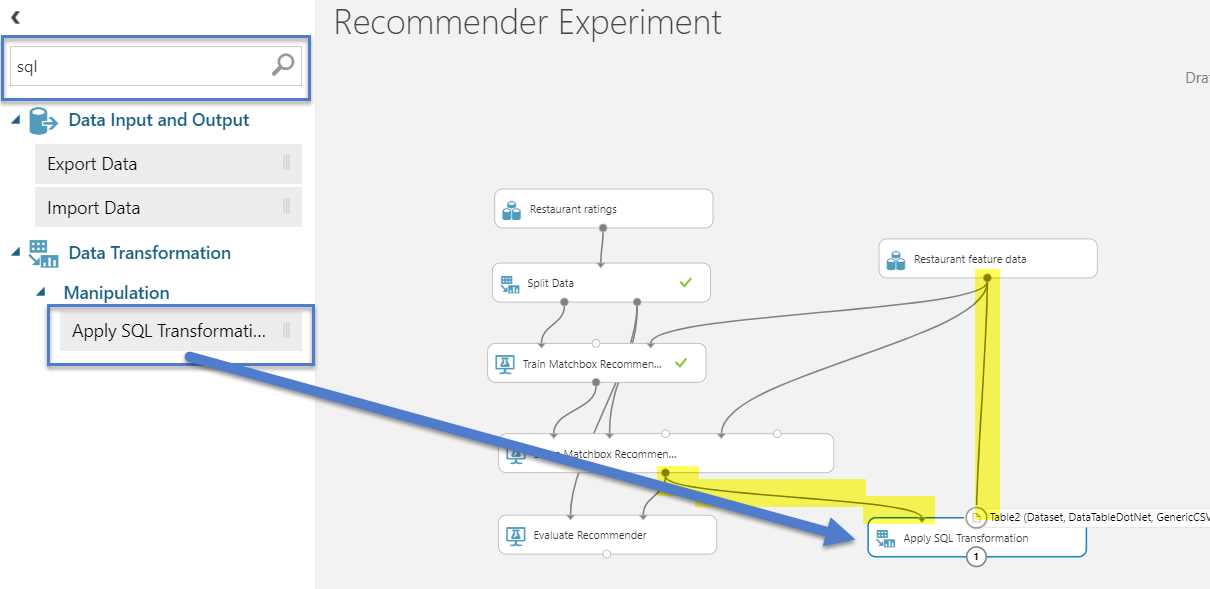
***Results....*** *(Aside: Metric "NDCG" = Normalitive Discounted Cumulative Gain)*



1. **Making the results more readable.** At this point we have a recommender which produces raw results, however, restaurants are just restaurant id's and users are user id's. Let's make one of those more readable: we will add "Restaurant Feature data" so that we can return the restaurant name rather than the a restaurant id. This is only half of what's required. Next, we'll use the "Apply SQL Transformation" to do the magic of converting ID's to Names.
   * Note that the Train and Score modules have input port for "Item Features" (in contrast with "User Features"). This allows us to pipe in dimensional attributes like restaurant names to be present in visualized results.
   * Wire the feature data into the Train and Score modules, as shown in yellow:

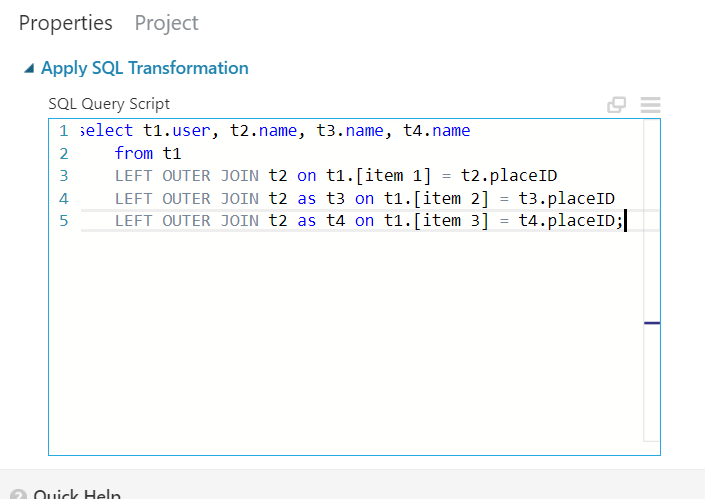


* **Converting IDs to Names using SQL Transformation module.** 
  + Add the Apply SQL Transfomation module and wire it in as shown

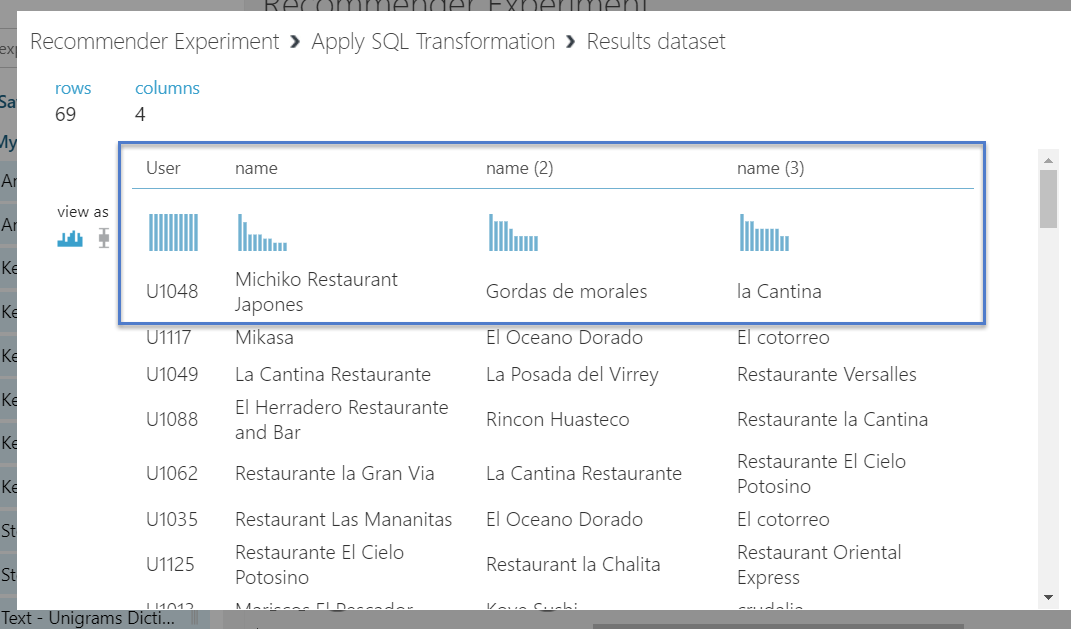


* Next, enter this SQL statement in the properties of the Apply SQL Transformation:

|  |
| --- |
| select t1.user, t2.name, t3.name, t4.name  from t1  LEFT OUTER JOIN t2 on t1.[item 1] = t2.placeID  LEFT OUTER JOIN t2 as t3 on t1.[item 2] = t3.placeID  LEFT OUTER JOIN t2 as t4 on t1.[item 3] = t4.placeID; |



* **Re-run the experiment** using the play button at the bottom of the screen.
* Right click | Visualize the output port on the Apply SQL Transformation and we see more readable results, as desired. We've also reduced the number of recommended restaurants.



# Section II: Create Web API

## Before you Start

Do Section I first so that you have created a Recommender experiment. Verify that it works within ML Studio. I.e. that it returns a list of restaurants for every user id. If you chose a different dataset (movies, products, etc) then the steps will virtually be the exact same.

## Concepts

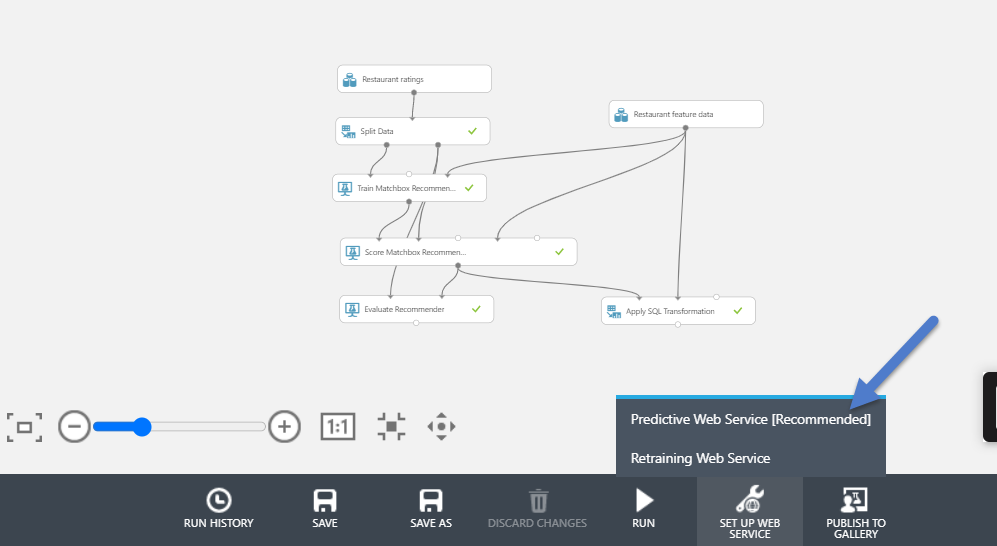
Creating a web api in ML Azure is easy.

We shall see in Section II that there are some additional steps to make it ready for a call because in Section I we returned the ***entire list*** of recommendations for ***every*** known user id, provided from the **static** **list of triples**. Now, we just want it to return data based on a single user id which will be provided dynamically (i.e. the user ID is not known to the predictive experiment until the web service provides it) .

Azure will also publish the web service which we will then be called in Section III from HTML using a Python based framework. The final end-to-end result from a webpage will be tested in Section III.

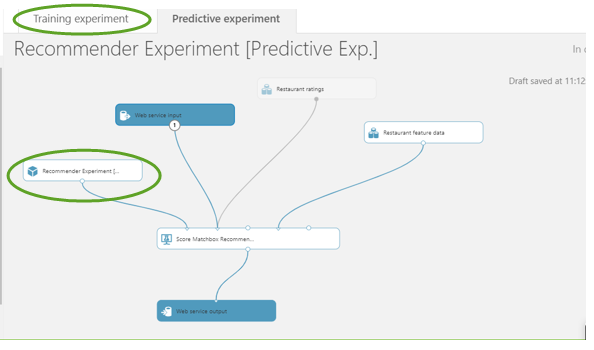
## Step-by-Step

1. Starting from your Experiment, click "**Set up Web Service**" from bottom of the page and sub-option "Predictive Web Service [Recommended]". When you do, you will see Azure animate the copy and transformation of your experiment to a "predictive experiment"

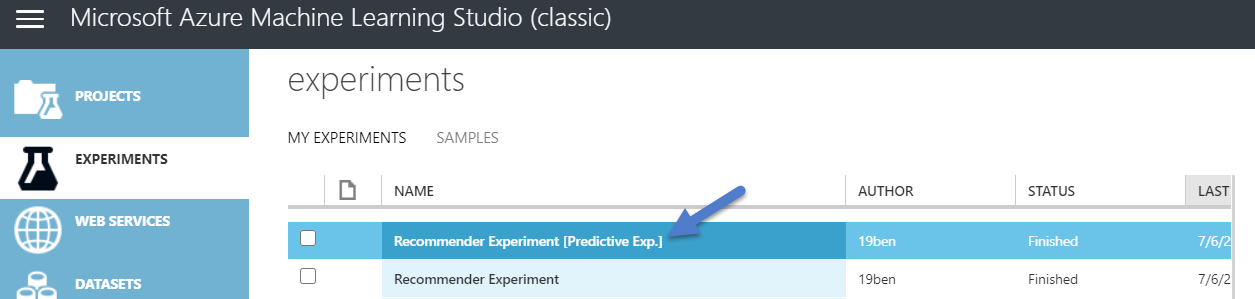


After clicking "Set up Web Service" you will observe a few things have changed in ML Studio:

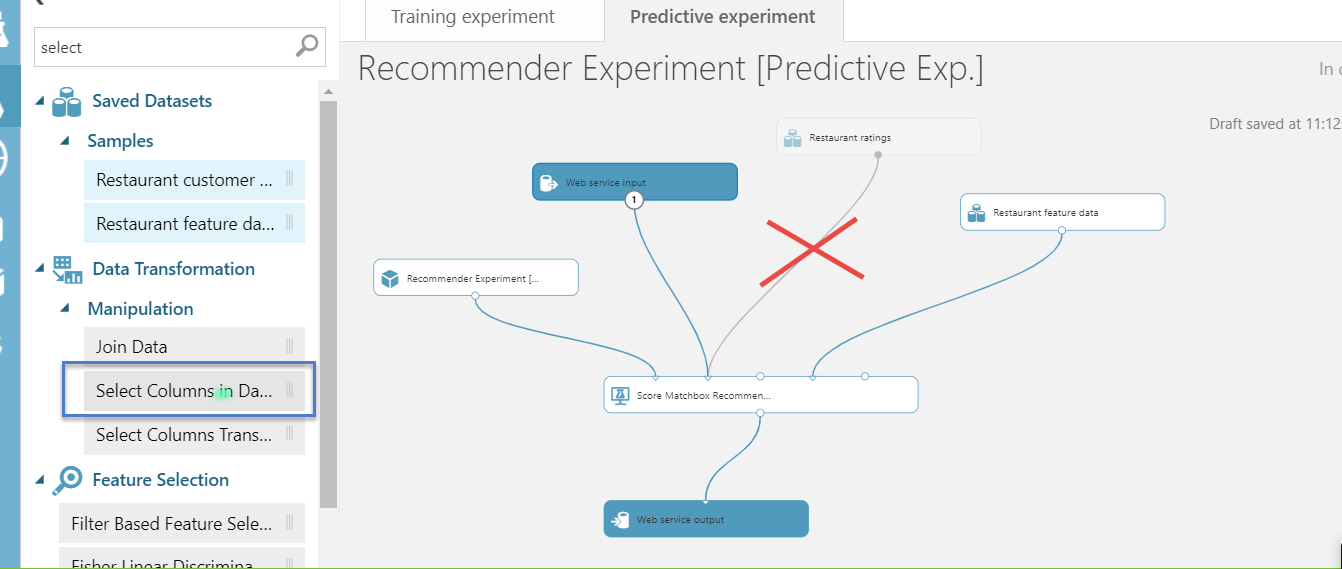
* You will now have two experiments: the original "Training Experiment" and the "Predictive Experiment". You will notice that the original experiment has been encapsulated in a "Recommender Experiment" module (shown with green circle) which is really ***just referencing everything we did in Section I***, which is now refered to as the **Training Experiment.**
* Also, the Predictive Experiment also has two new modules in blue for "web service input" and "web service output."
  + *Looking ahead:* We want our web service to accept an input of "UserID" (in the real world, this could be a part of town you're in or a user id if you're logged into a service like Yelp) and to return output of three (3) top restaurant recommendations.



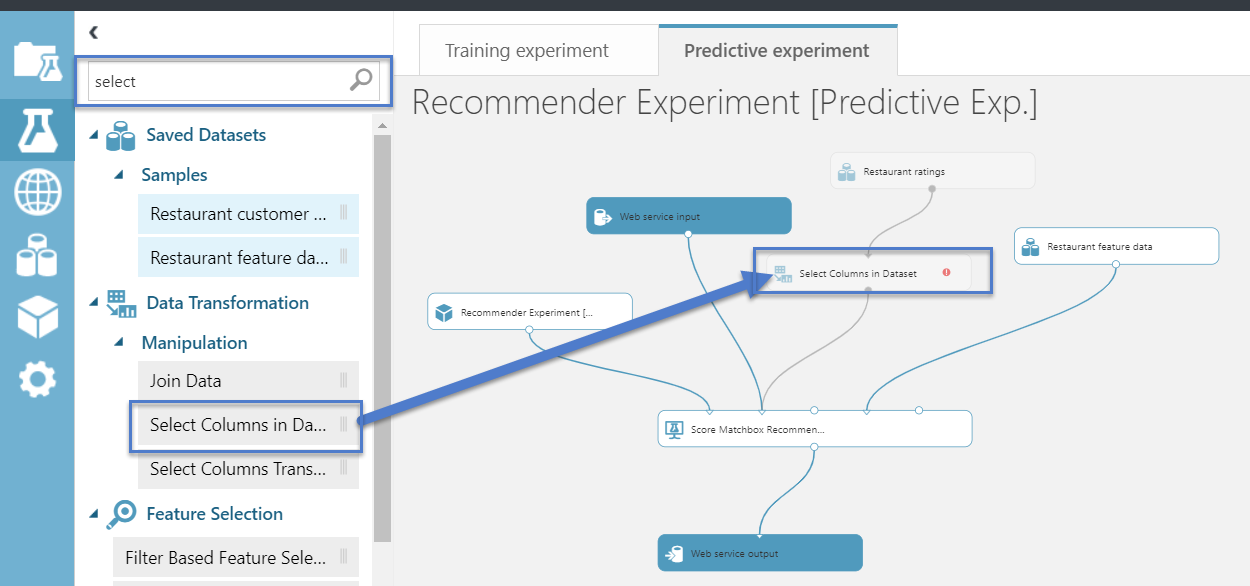
* To belabour the point, Azure has created a second experiment on your behalf which the web service will call to perform net-new predictions. You don't need to navigate to it as you are already inside the predictive experiment.



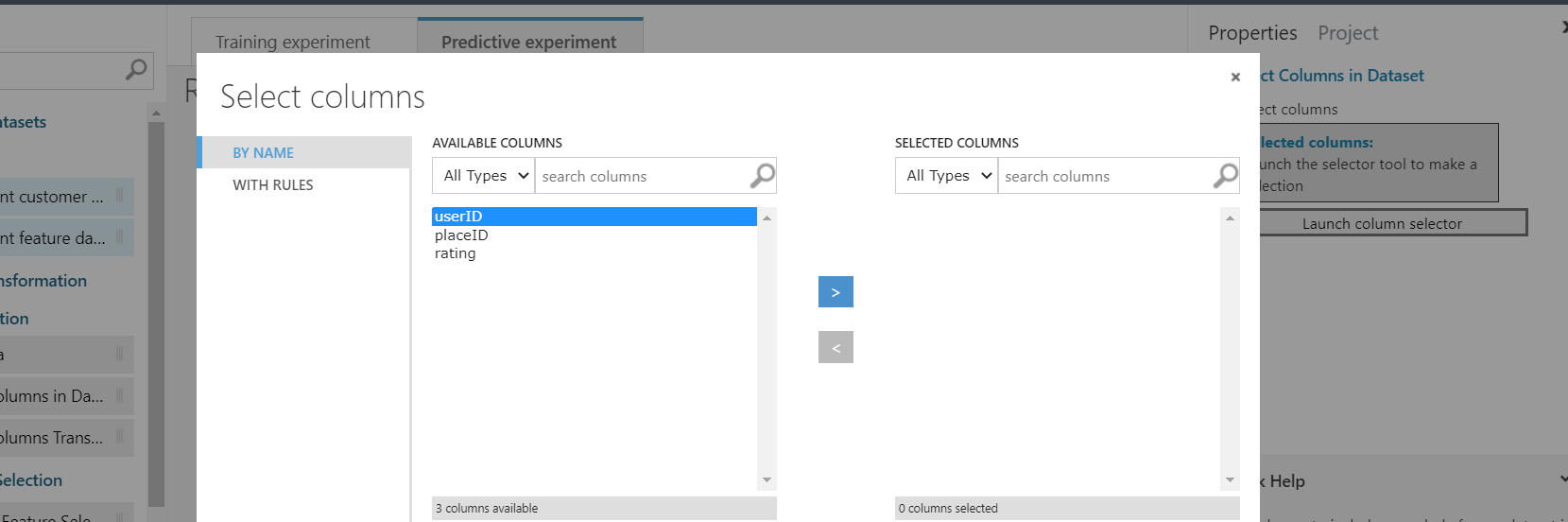
1. Remove the link between *Restaurant Ratings* dataset module and the *Score Matchbox Recommendation* module.
   * Note: We will be dragging in a new module bewteen them in the next step. Hang tight.

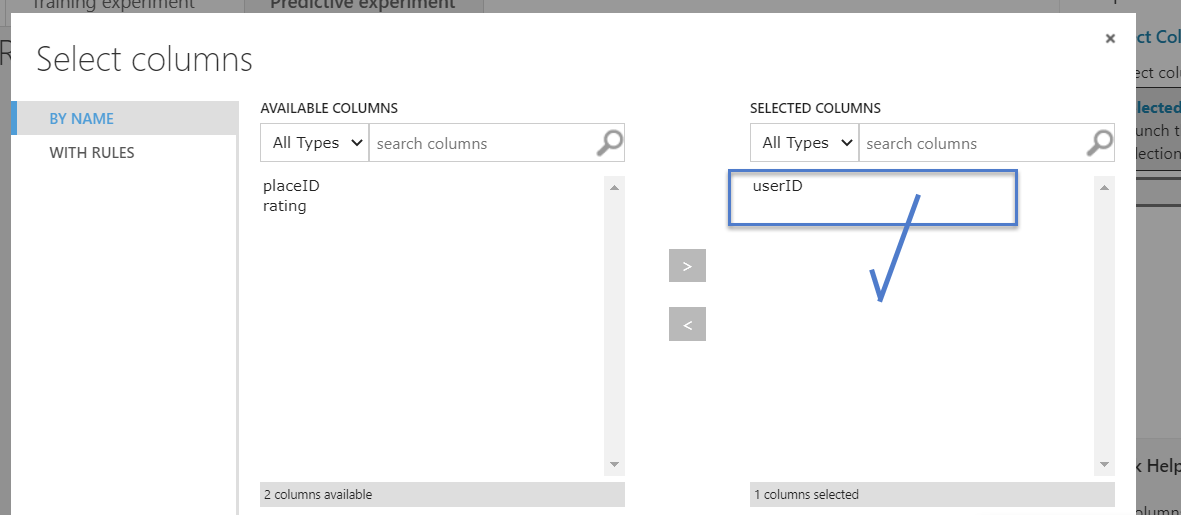


1. Drag in a "*Select Columns in Dataset*" module.
   * "Wire it in" by dragging *Restaurant Ratings'* output port to *Select Columns in Dataset'*s input port. Then, drag *Select Columns in Dataset's* output port to *Score Matchbox Recommender's* input port, as shown.

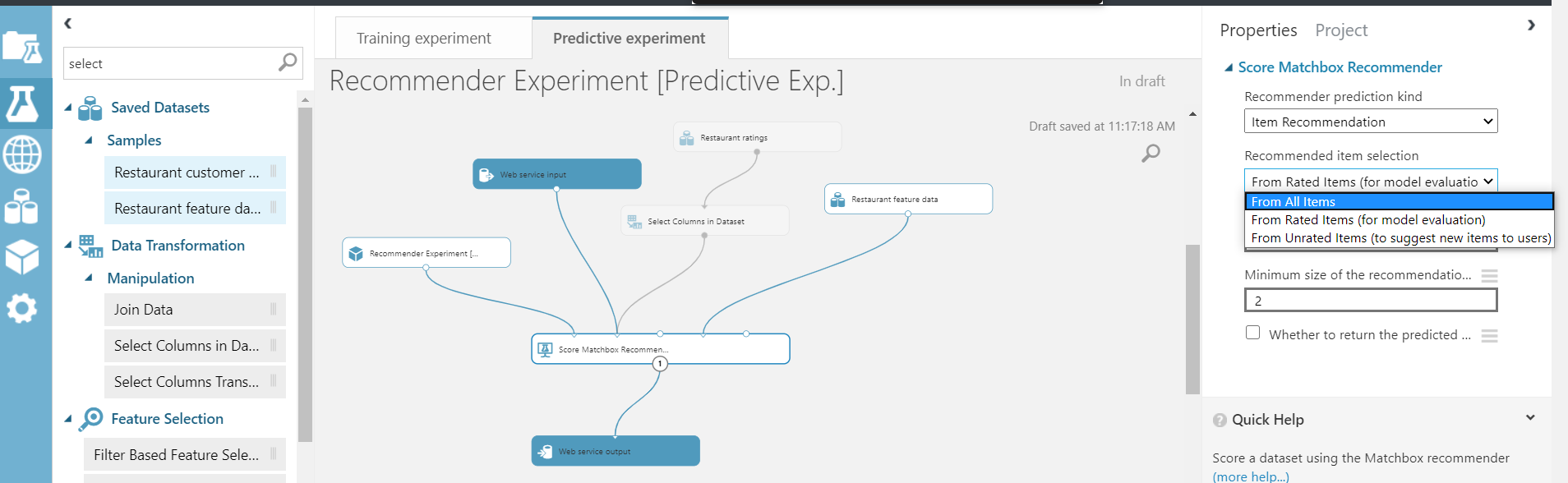


1. Configure the "Select Columns in Dataset" module:
   * Select UserID so that it shows under Selected Columns, as shown.

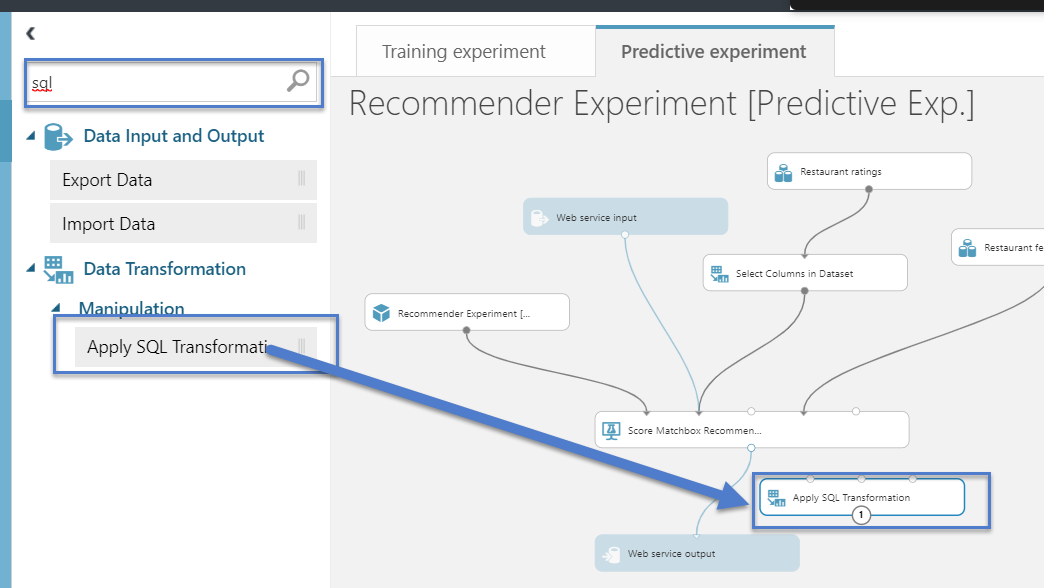




1. Next, we configure the *Score Matchbox Recommender* module to change its property from "From Rated Items (for model evaluation)" to "From All Items". Remember to select the module first so that its properties show in the right-hand panel.

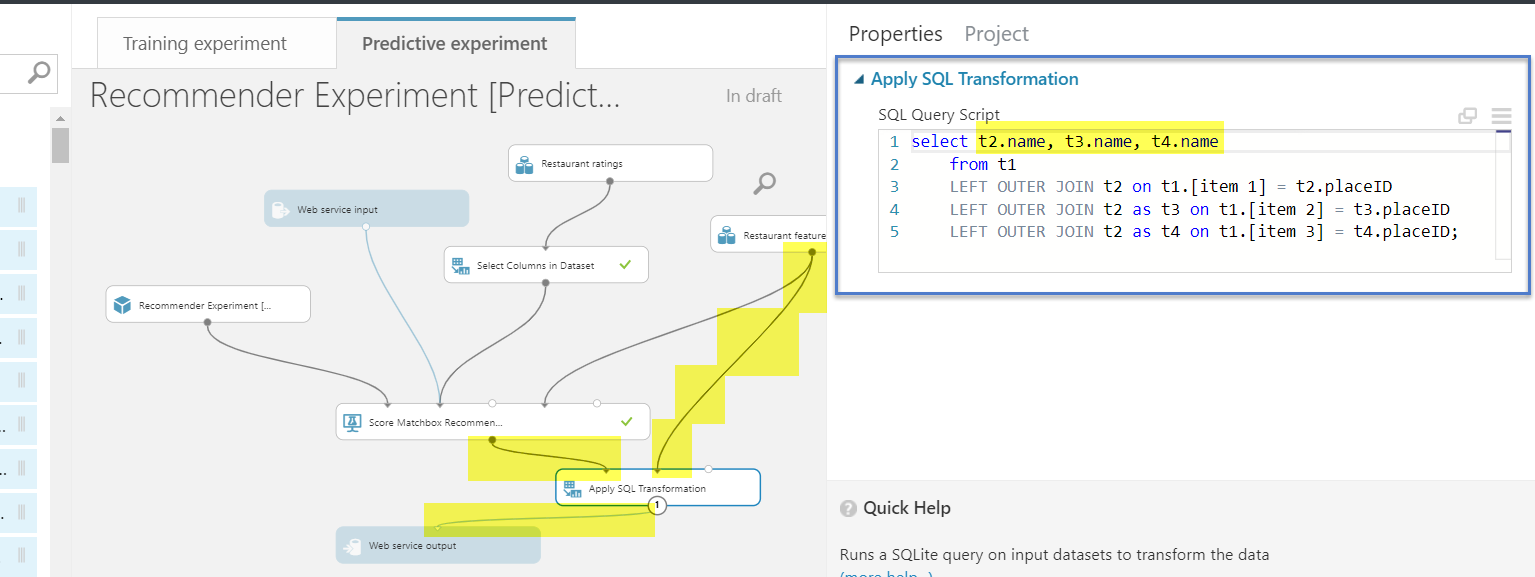


1. Drag in an *Apply SQL Transformation* module between the last two modules.

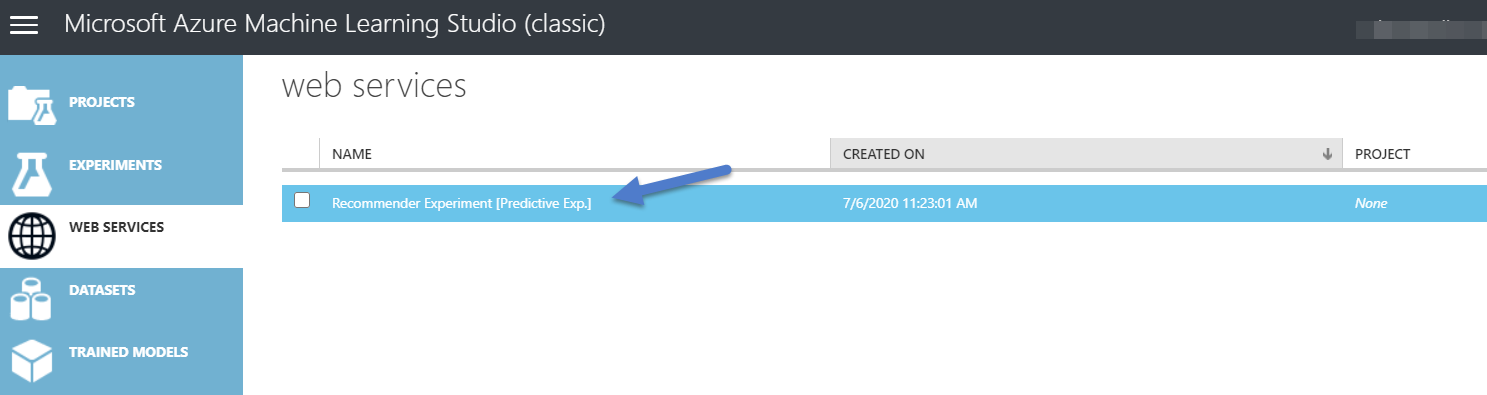


1. Wire-in and then configure the properties on the new Apply SQL Transformation module, as shown.
   * The SQL command is almost the same as the one we saw in Section I except it doesnt include userID. Since we'll provide UserId as an input from the web service input we dont need to see it in our output, which is why it's not required here.

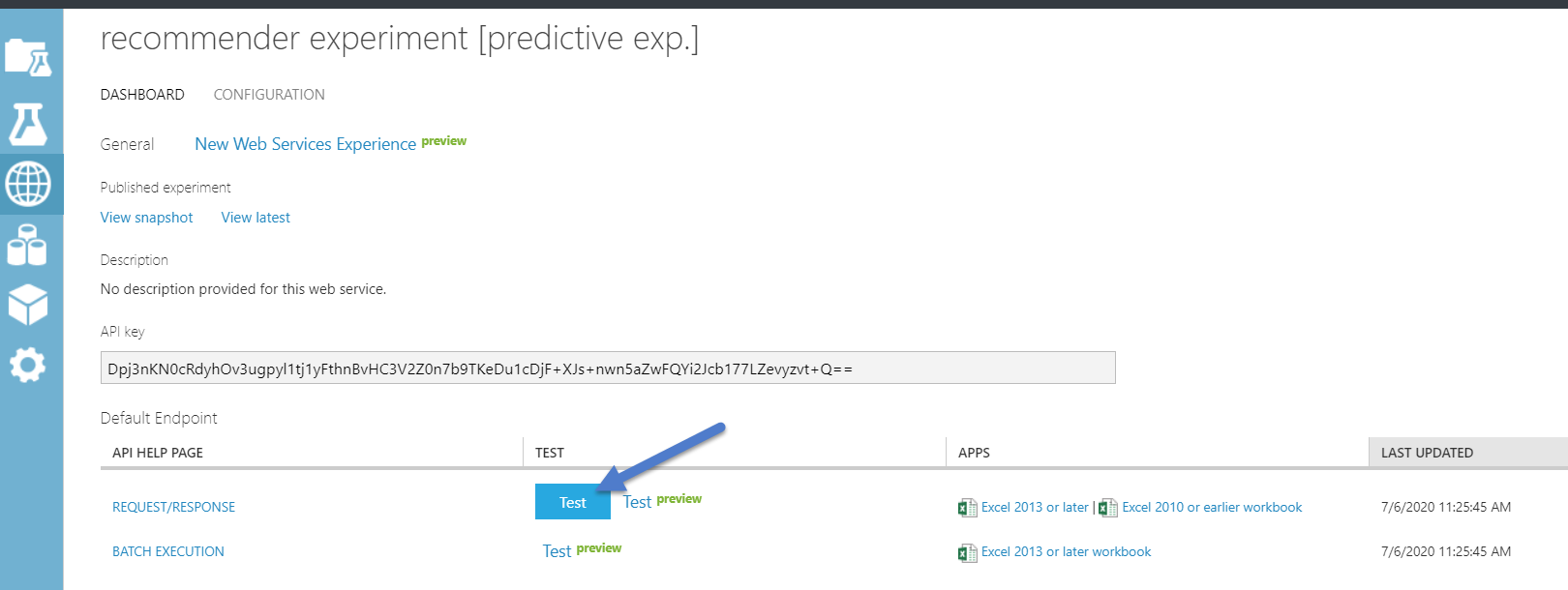
|  |
| --- |
| SELECT t2.name, t3.name, t4.name  FROM t1  LEFT OUTER JOIN t2 on t1.[item 1] = t2.placeID  LEFT OUTER JOIN t2 as t3 on t1.[item 2] = t3.placeID  LEFT OUTER JOIN t2 as t4 on t1.[item 3] = t4.placeID; |

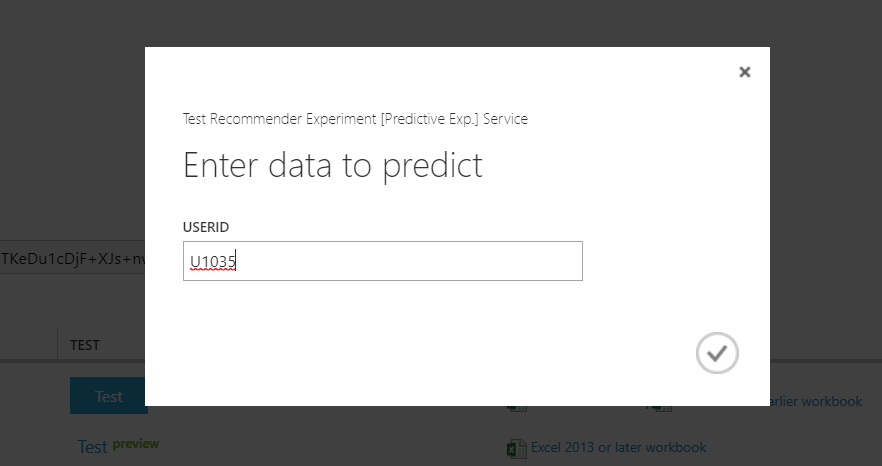


1. At this point, our Predictive Experiment is set up and Azure has published a web service which we will now observe and test in the next step.
   * Go into "Web Services" and select your "Predictive Experiment's" web service.

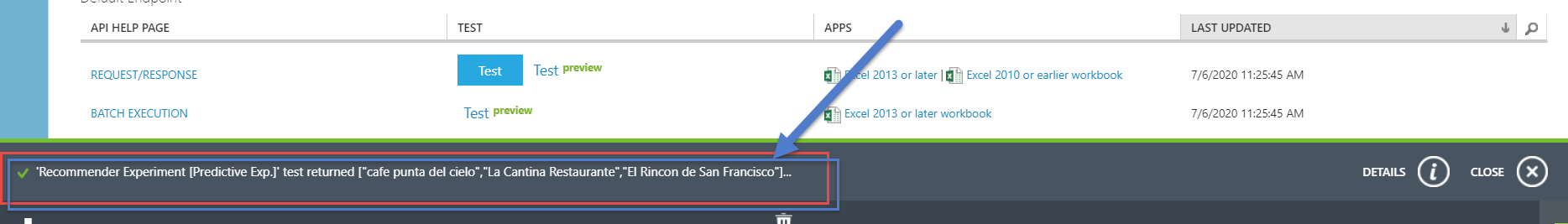


1. **Unit Testing the new web API in ML Studio.** 
   * Click "Test" button on the "Request/Response" line. Enter "U1035" as the UserID. This is just one of the Users in the original dataset.





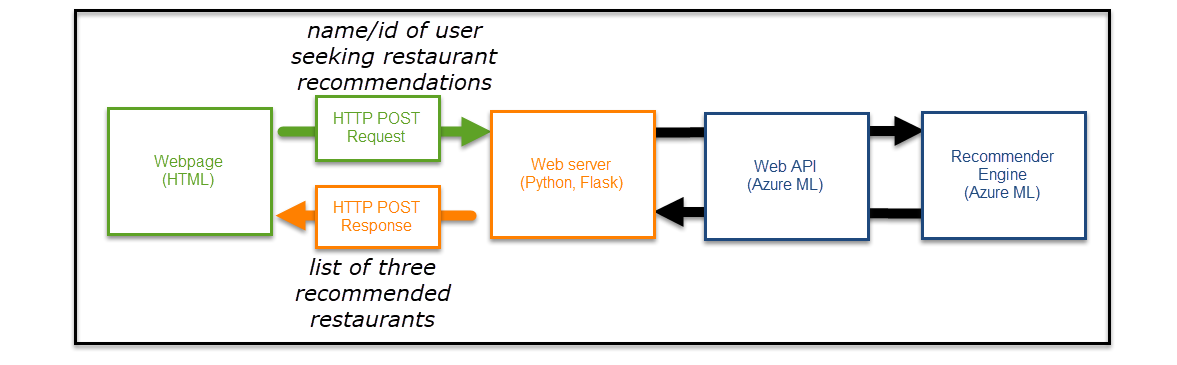
* The expected, returned result is shown at the bottom of the page in ML Studio. Note the columns that are returned are those we specified in the Apply SQL Transformation module.
  + **This concludes our unit test of the web API.**



# Section III: Reference Web API

## Concepts

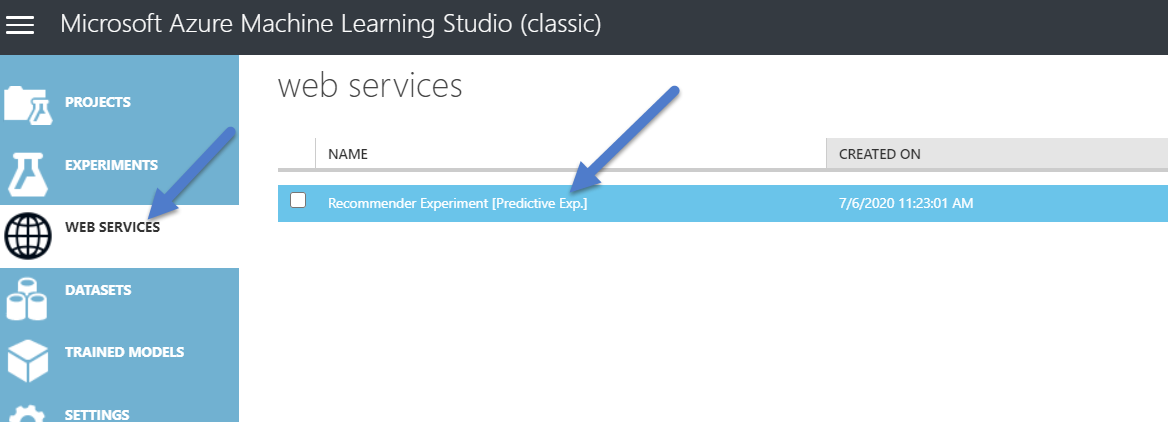
We now get to the good stuff: we will see this recommender working from end to end, from a webpage, thru the API call, to the recommender system.



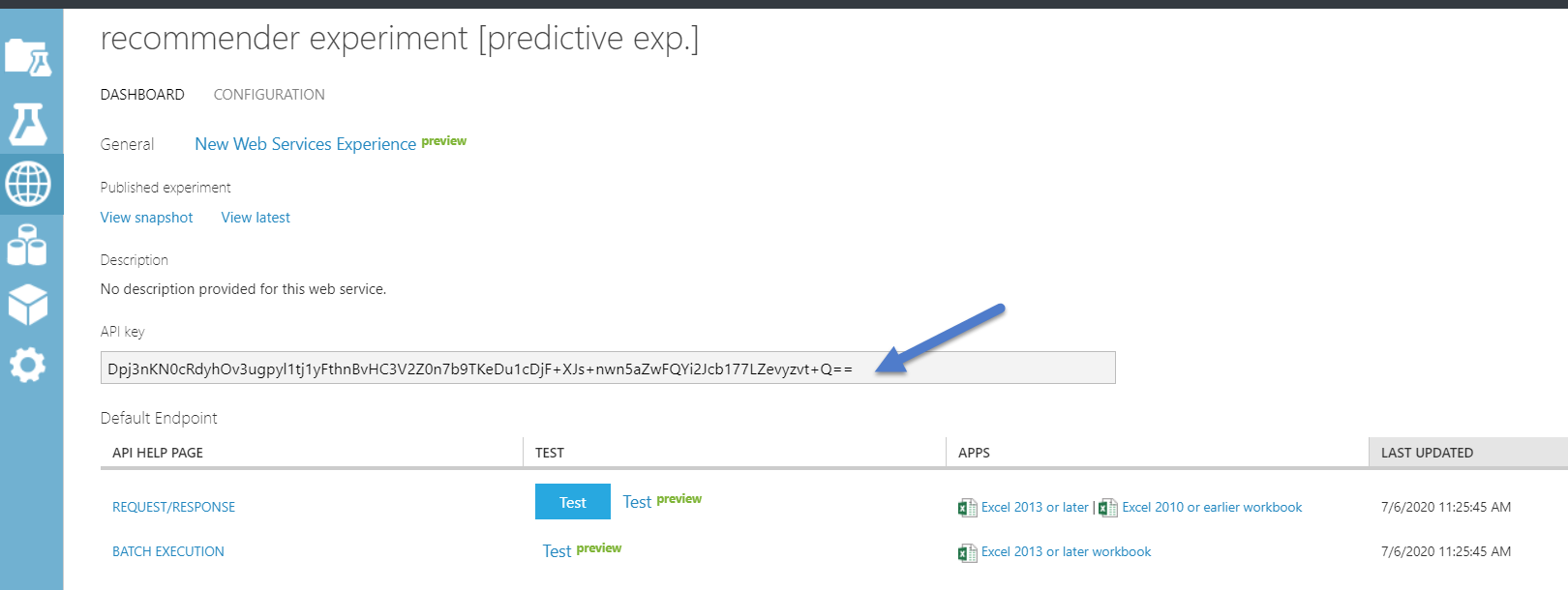
## Step-by-Step

1. Login to Azure Machine Learning Studio(classic): <https://studio.azureml.net/>

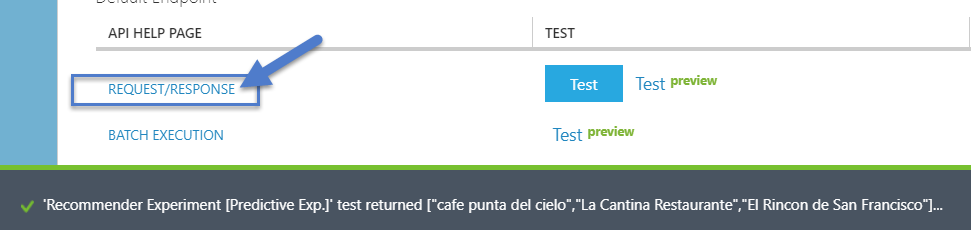
* Within Azure ML Studio, click ***WEB SERVICE*** from the left menu bar and select the deployed web service for the Predictive Experiment we tested in Section II.



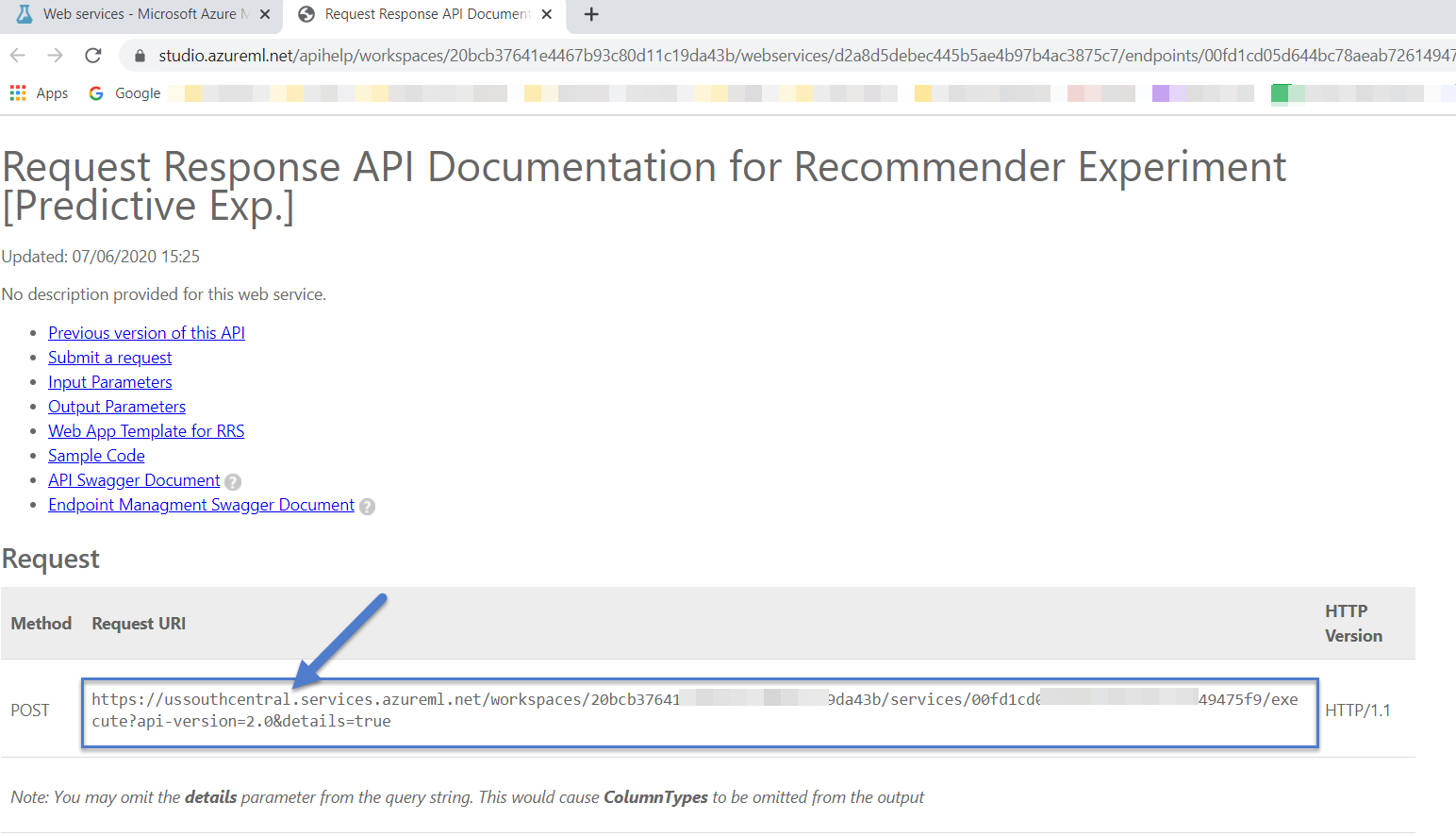
1. **Copy the API Key so we can construct a call to our new API from *outside* Azure.**
   * Copy the API key to a temporary text file on your machine.



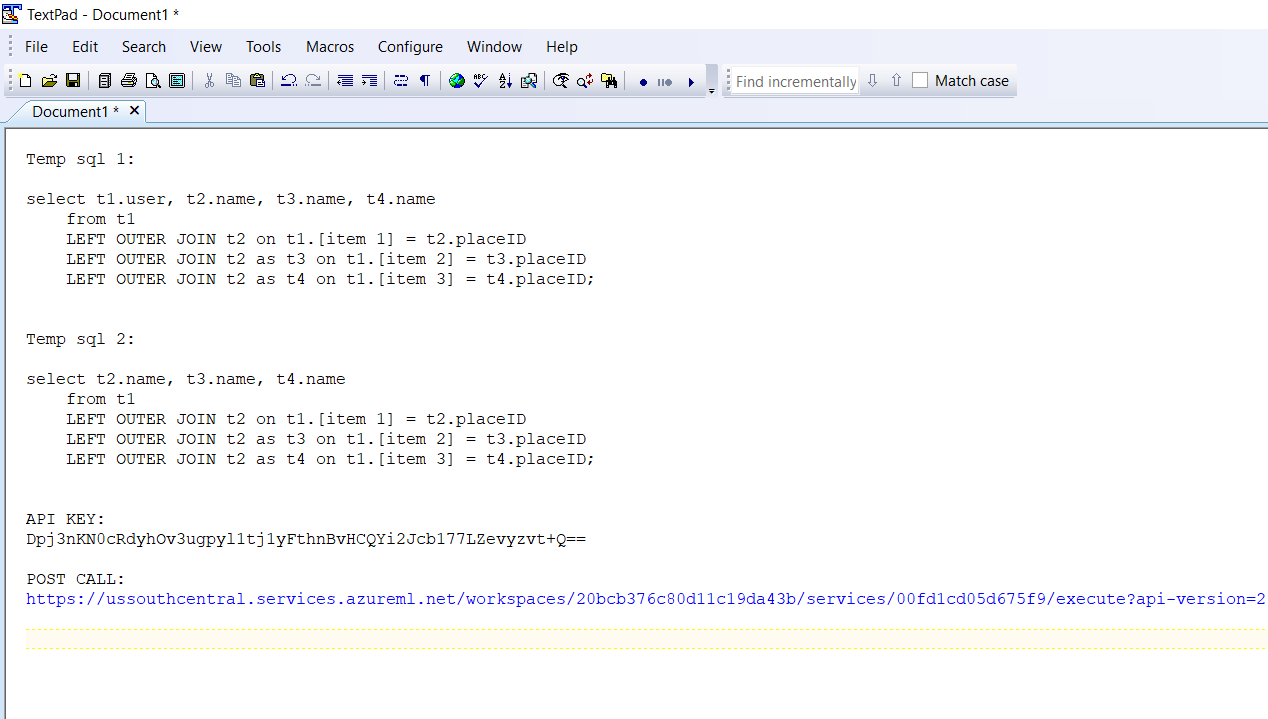
1. Copy the HTTP **POST RequestUI** call within the *Request/Response* hyperlink.
   * Click “REQUEST/RESPONSE” on the *Web Services* panel.



* + Copy the **Request URI** into a temporary text document, like the one where you put the API Key. We will use both the API Key and the Request URI (~same thing as a URL), shortly.



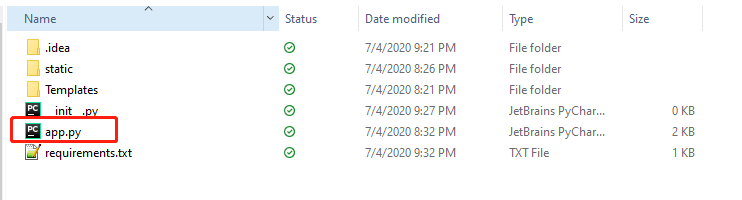
* + Snapshot of a temporary text file:



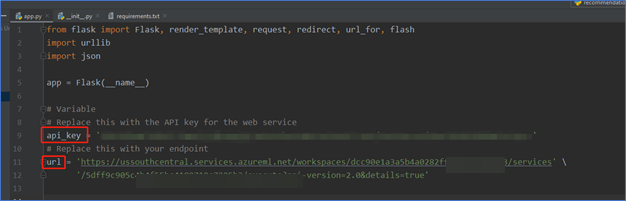
1. Extract *“restaurant recommendation.zip”*. It is also available at Github.
   * The \*.zip file is embedded here:



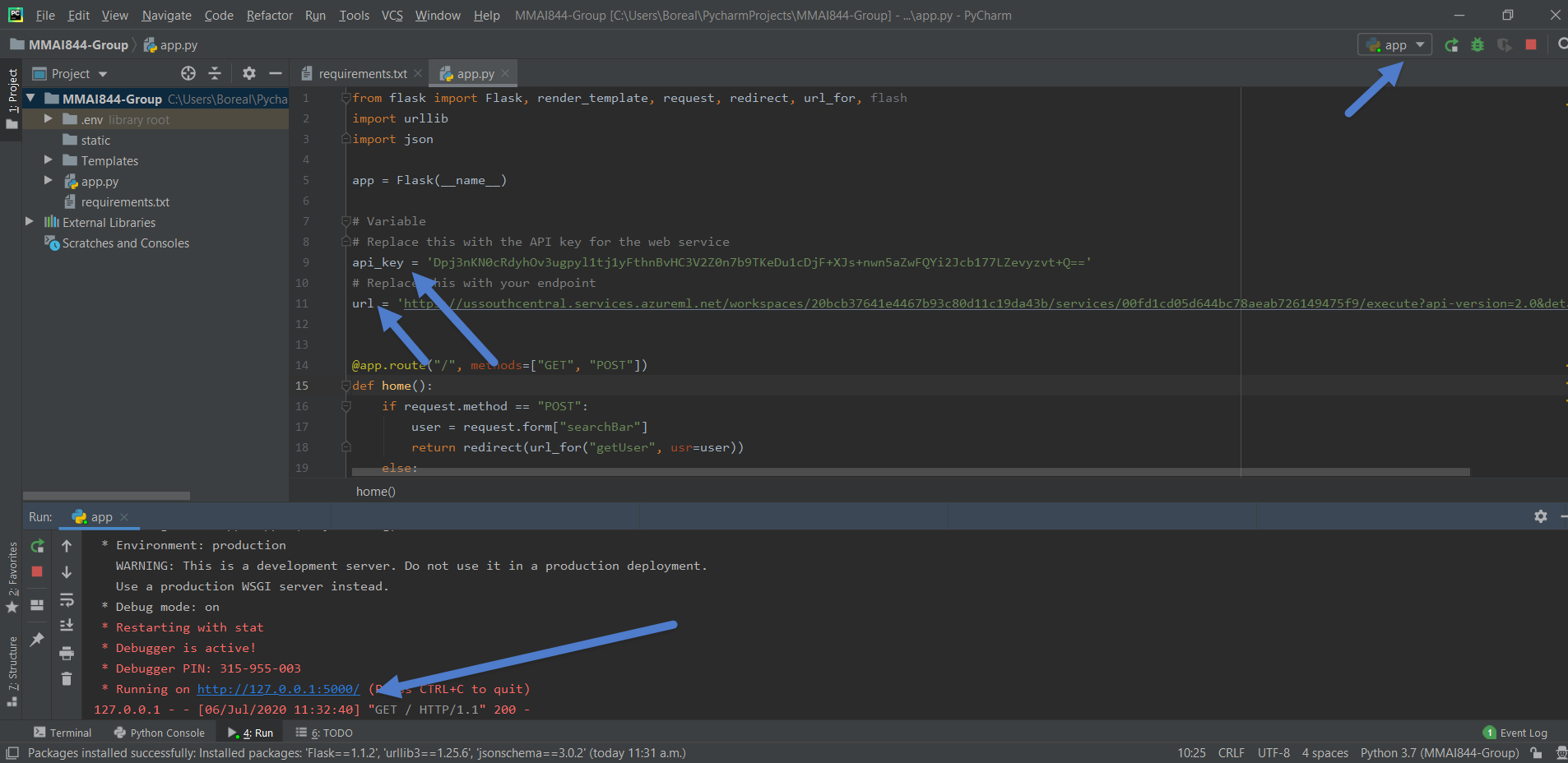
* + Please open Python file **app.py**.
    - Please make sure all the required packages are installed on your laptop – see the required Python libraries listed in the *requirements.txt* file.



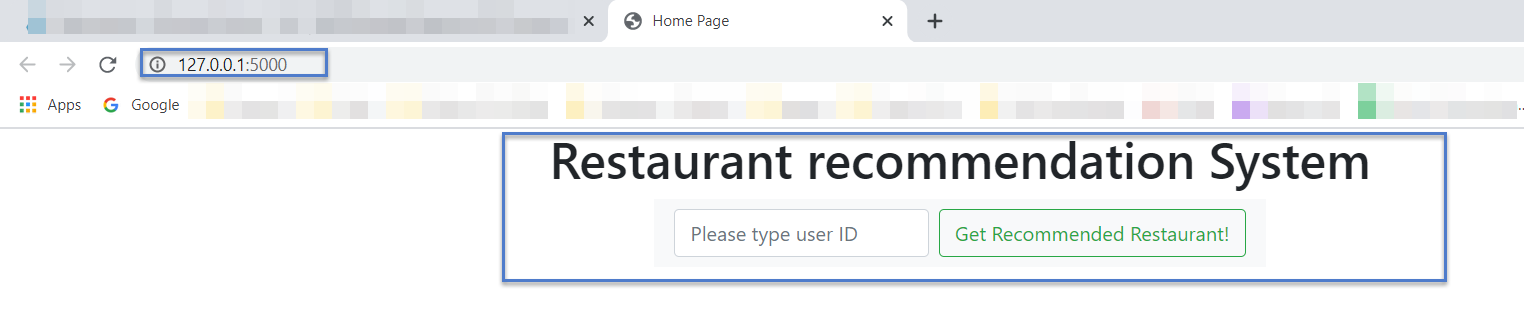
1. There are two Python variables in this file that need to be replaced.
   1. **api\_key** – please replace this one with your personalized **API key**
   2. **url** – please replace this one with your stored **Request URI**
      * *Some of the data in the snapshot below has been obscured for data privacy.*



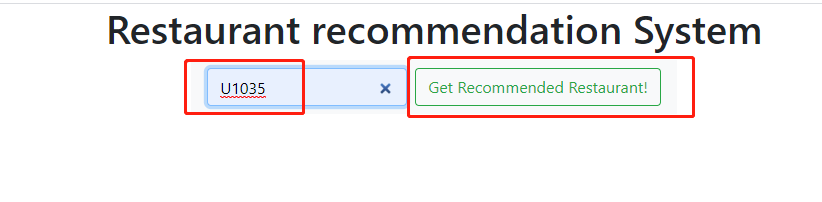
1. Save the Python file with your changes made to **api\_key** and **api.**
   * **Run the Python script.**  *The snapshot below uses the Pycharm IDE though any IDE will work.*
   * In effect, Python uses "*Flask*" which creates a local web server and which could be plausibly embedded in a formal production webpage as part of a feature within a larger ecosystem.
   * **Python (Pycharm IDE) will indicate that the web server is running.**
     + As default, the application will be running on <http://localhost:5000>
     + An *equivalent* URL you might see and can also click is: <http://127.0.0.1:5000/>



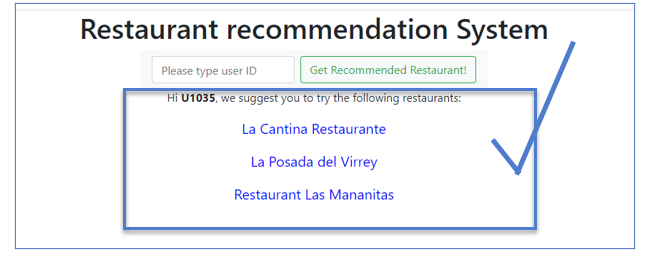
* You should see this initial webpage when you click on the web links, above. The webpage is ready to accept input.

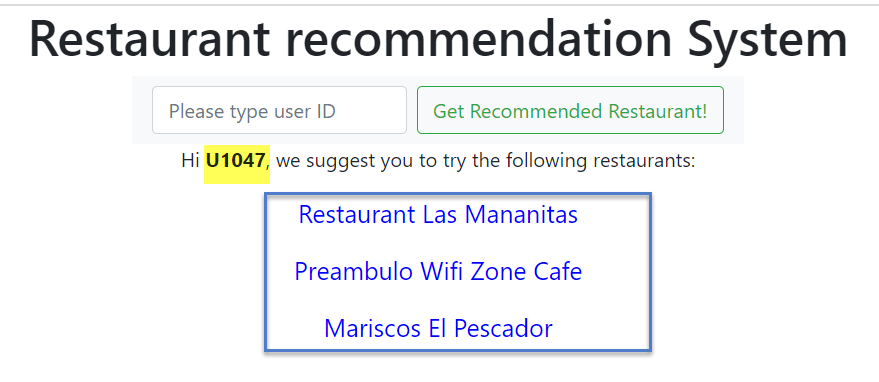


1. **Test the Web API through a webpage (that is, from a place that is external to Azure.)**
   * You can input any user ID in the input box(e.g: U1035), and please click “Get Recommended Restaurant!”
     + If you enter "Johnny" or any other arbitrary name you will get default recommendations that are three very popular restaurants.



1. **Congratulations...!**
   * If you see the following web page, congratulations, your web service is up and running!
   * The three restaurants are personalized recommendations for the user shown. Trying it for a different user, say User **U1047**, you will see that it yields a different result, as expected.





1. **To review,** in this training document you have:
   * set up and trained a recommender system,
   * created an API with which to call the recommender system,
   * created and tested a webpage which calls the API with the input, and which displays the returned results as a dynamically-created webpage.

**Final result:**

