Introduction to Machine Learning on IBM Watson Studio

Upkar Lidder

Developer Advocate, IBM

> ulidder@us.ibm.com
> @lidderupk
> upkar.dev

IBM Developer

http://bit.ly/waston-ml-sign

Prerequisites

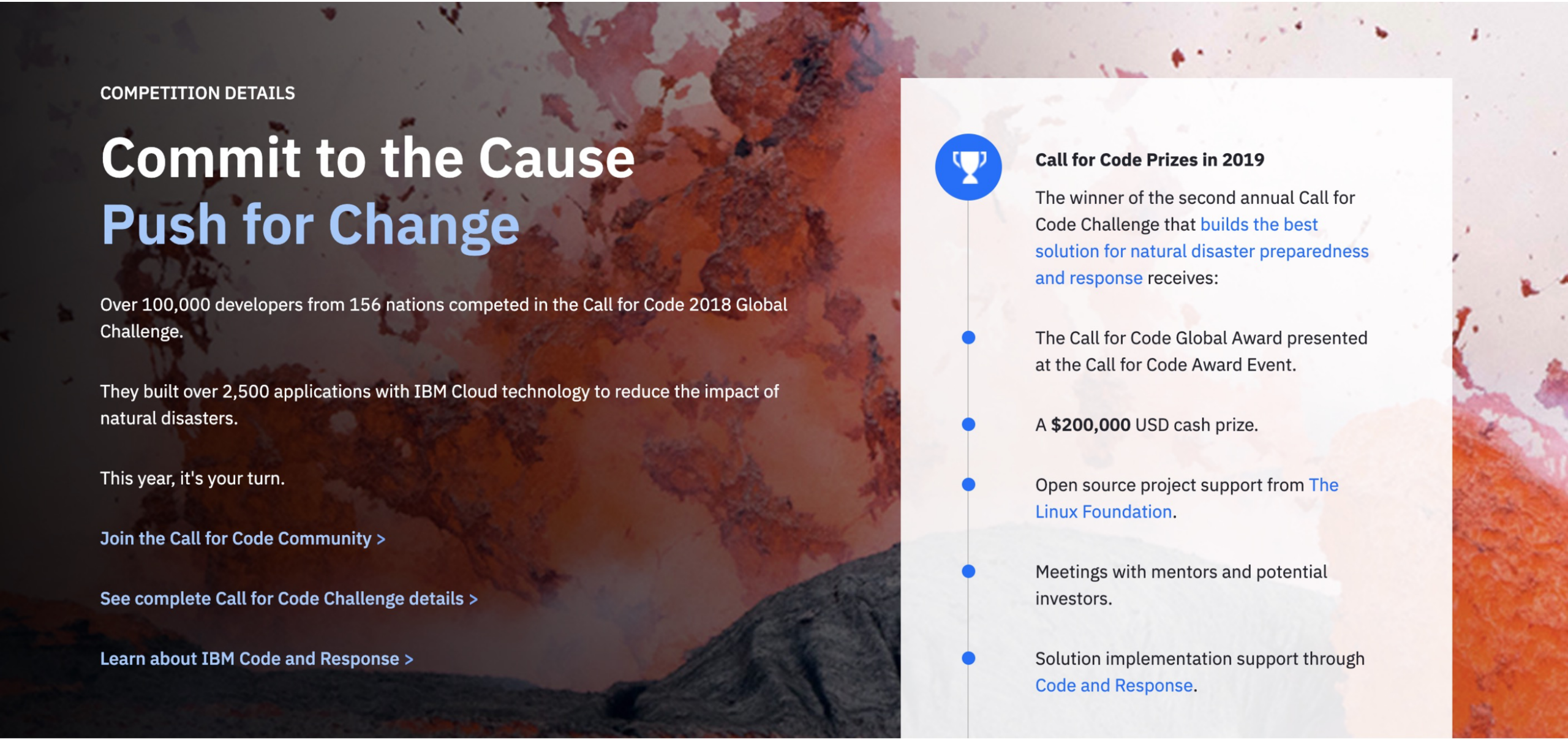
1. Create IBM Cloud Account using THIS URL

http://bit.ly/waston-ml-sign

2. Check your email and activate your account. Once activated, log back into your IBM Cloud account using the link above.

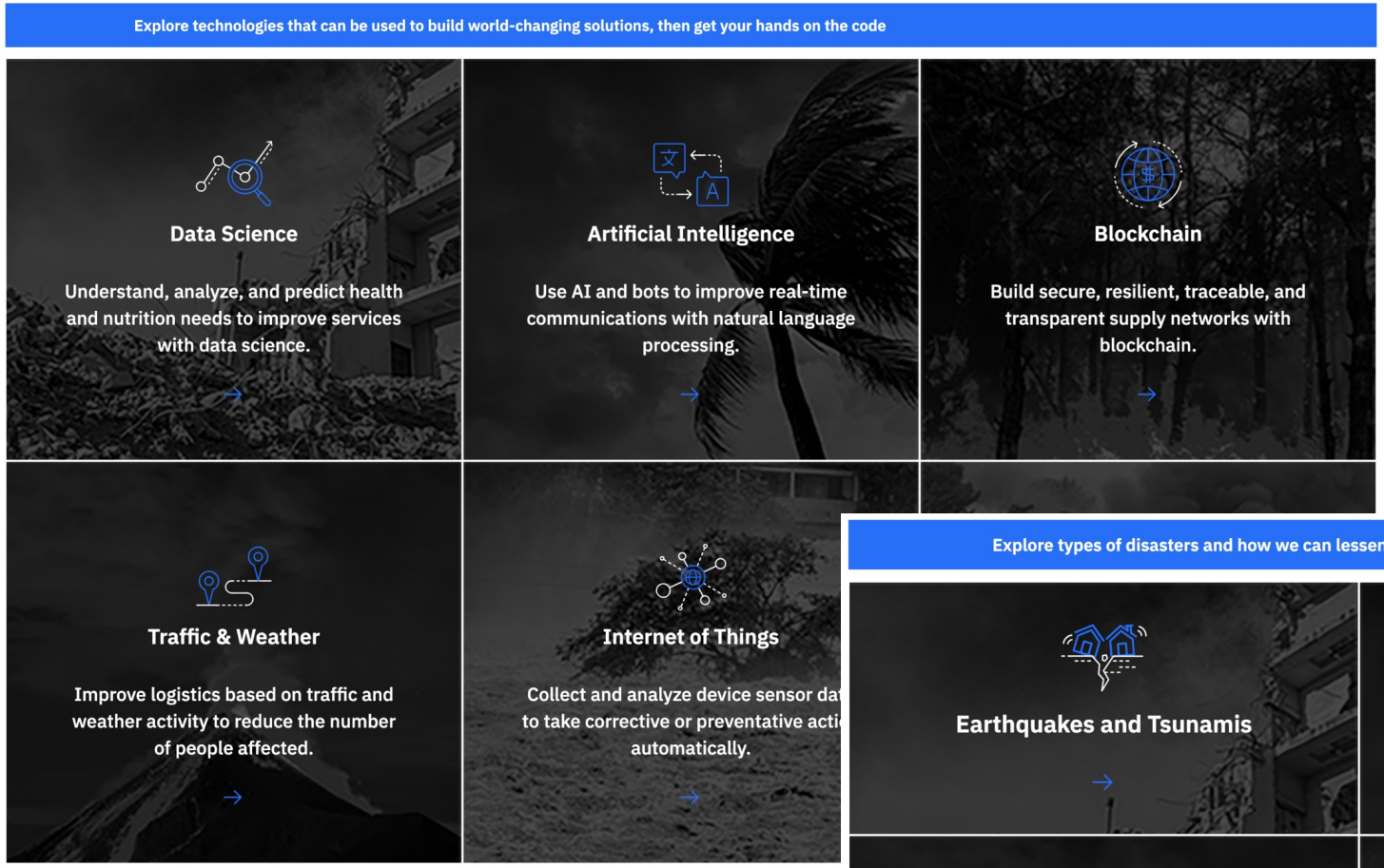
3. If you already have an account, use the above URL to sign into your IBM Cloud account.

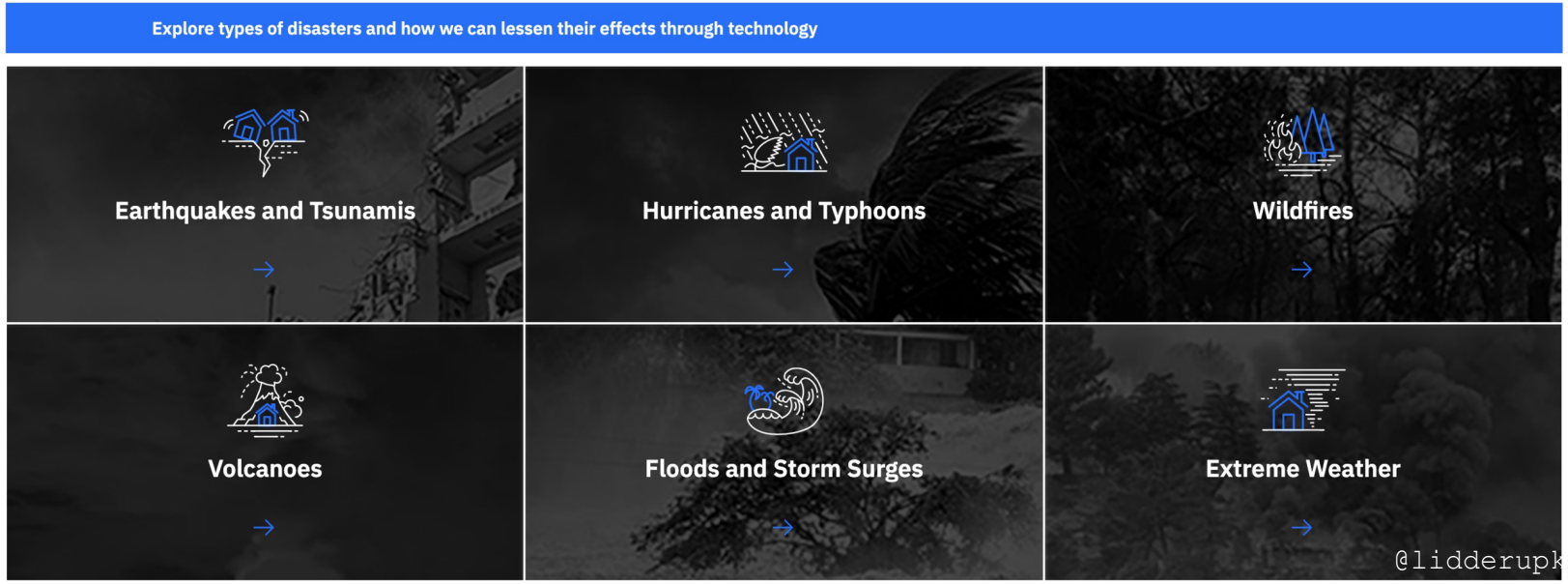
Call for Code 2019



Call for Code 2019 - Get started quickly

https://developer.ibm.com/callforcode/

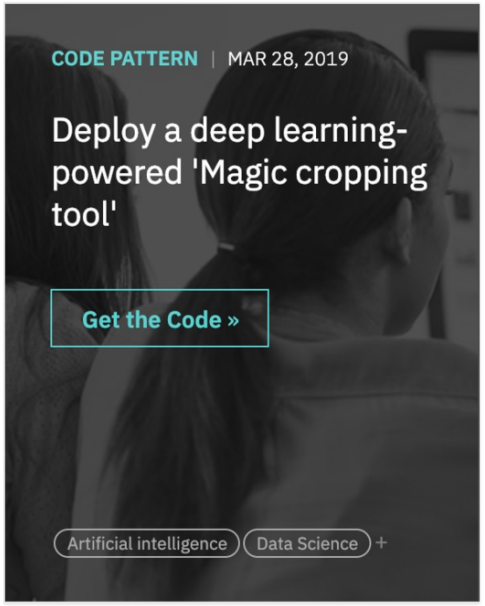


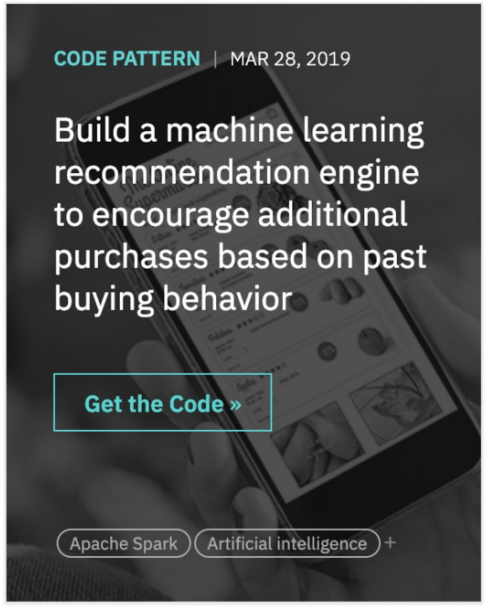


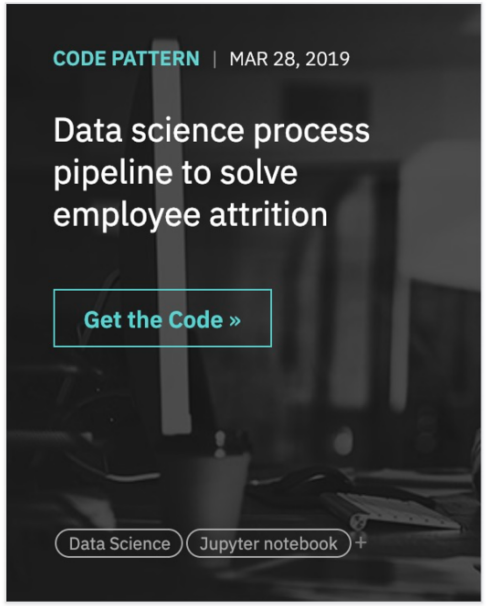
- Background
- Research
- Data
- Videos
- Solutions
- Blogs/Articles
- Code patterns

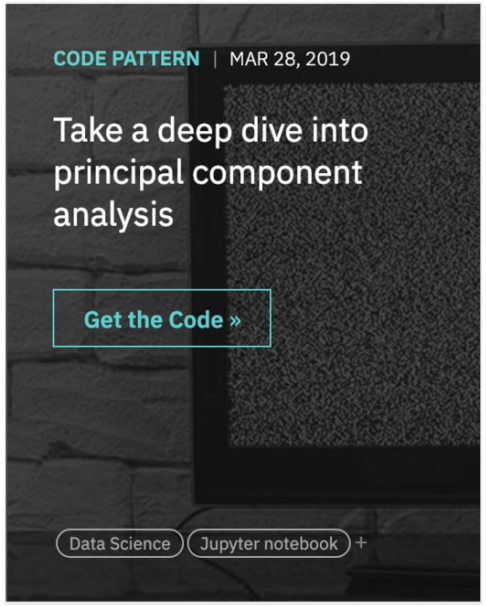
IBM Code Patterns

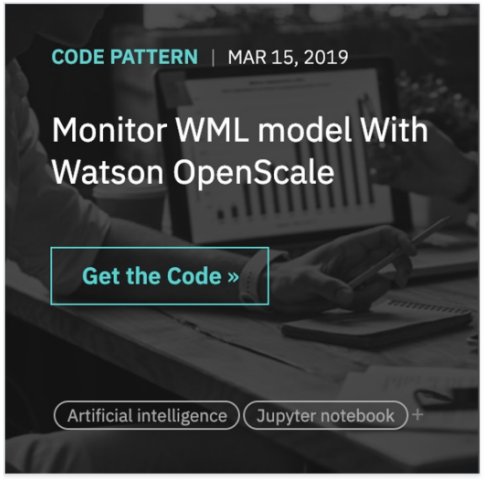
Machine Learning ×

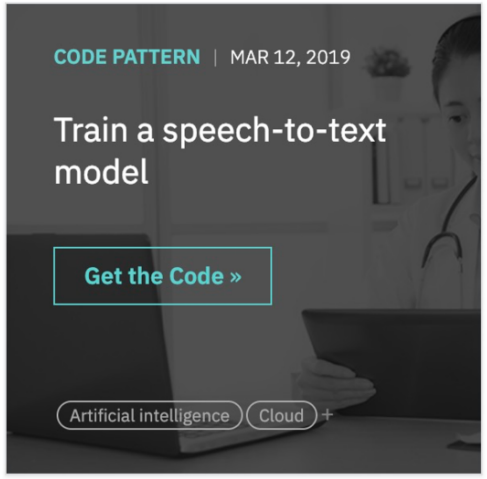


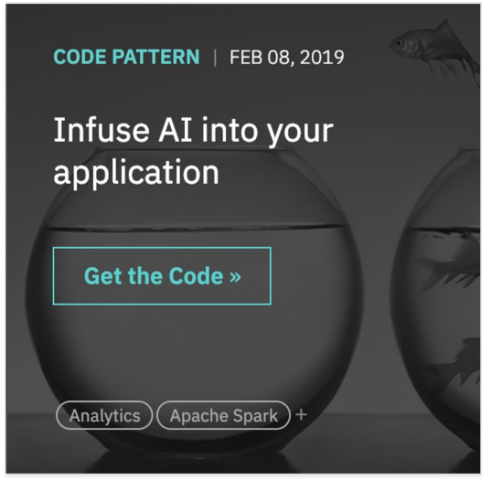




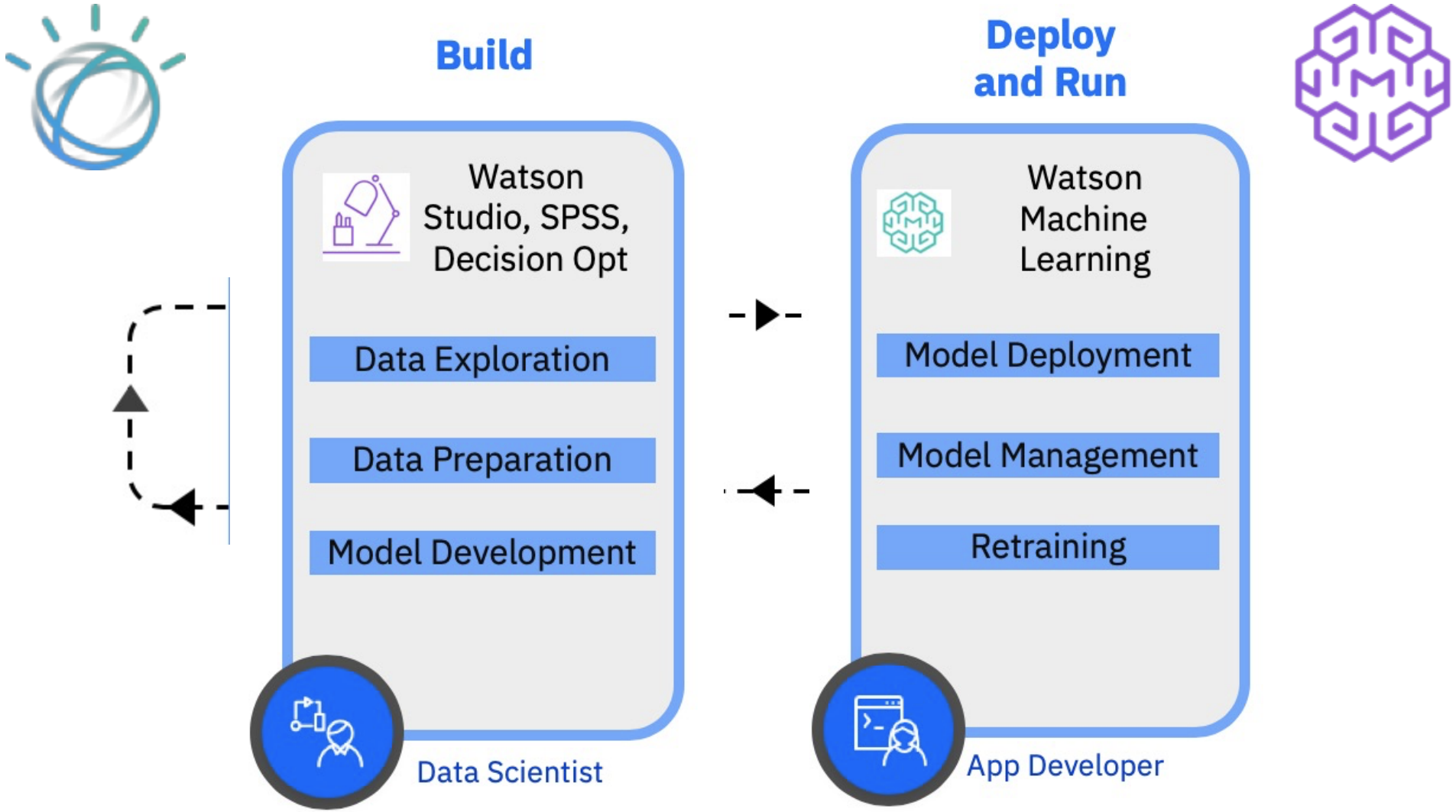








Watson Studio & Watson Machine Learning



Watson Studio

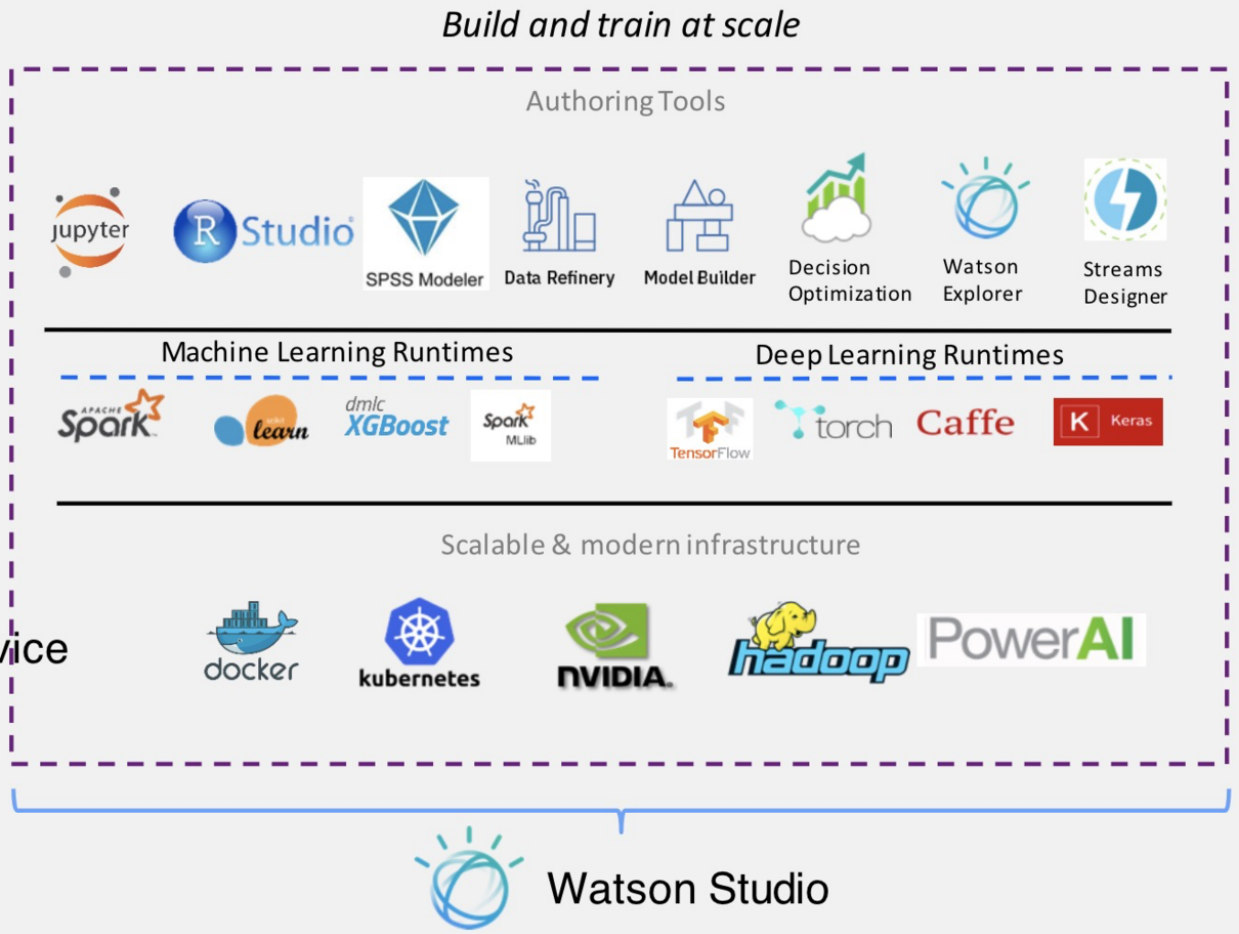
IBM Watson Studio

- Create, collaborate, govern and integrate
- Using best of breed - Open source & IBM tools
- Code (R, Python or Scala) and no-code/visual modeling tools

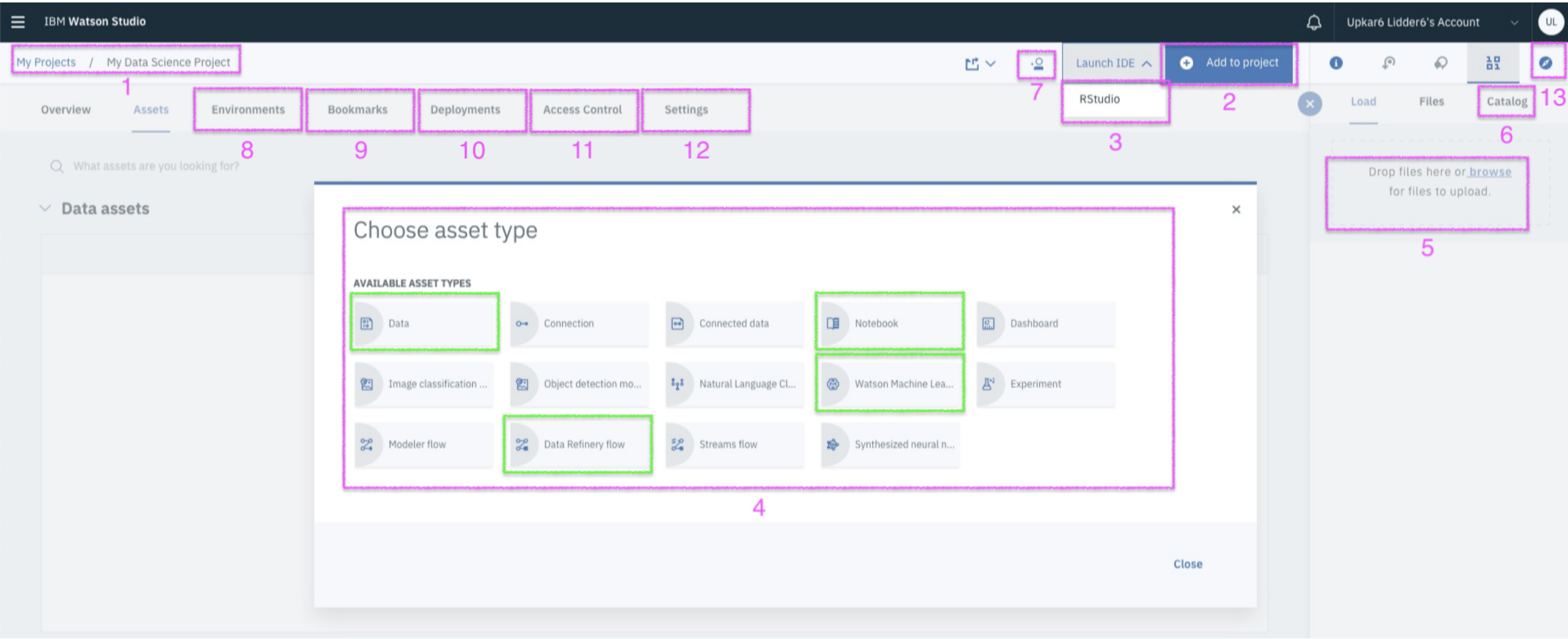
- Most popular open source frameworks
- IBM best-in-class frameworks
- Workflow driven data science

- Container-based resource management
- Elastic pay as you go cpu/gpu power
- Run on x86, Power, zLinux
- Integrate with Cloudera and HDP using Hadoop Integration service

- Train and deploy where your data lives
  - As a Fully managed Service
  - In your Data center
  - On AWS, Azure, IBM Soft layer



IBM Watson Studio



Workshop - Goals

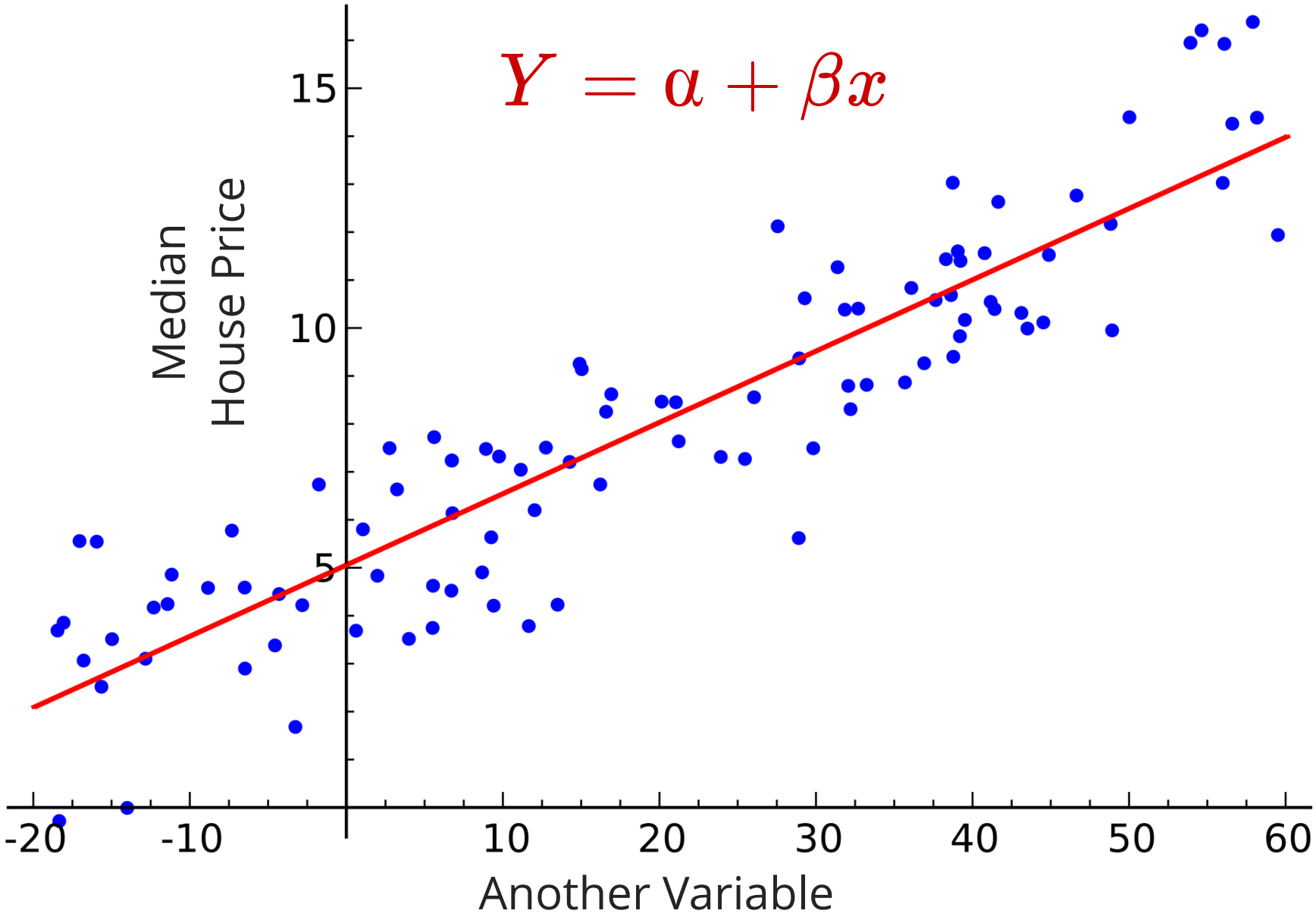
Successfully **Create**, **Store** and **Deploy** a Linear Regression Model on IBM Cloud using Watson Studio and Watson Machine Learning Services.

Question - predict median house price for Boston area

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.
```

	0	1	2	3	4
CRIM	0.00632	0.02731	0.02729	0.03237	0.06905
ZN	18.00000	0.00000	0.00000	0.00000	0.00000
INDUS	2.31000	7.07000	7.07000	2.18000	2.18000
CHAS	0.00000	0.00000	0.00000	0.00000	0.00000
NOX	0.53800	0.46900	0.46900	0.45800	0.45800
RM	6.57500	6.42100	7.18500	6.99800	7.14700
AGE	65.20000	78.90000	61.10000	45.80000	54.20000
DIS	4.09000	4.96710	4.96710	6.06220	6.06220
RAD	1.00000	2.00000	2.00000	3.00000	3.00000
TAX	296.00000	242.00000	242.00000	222.00000	222.00000
PTRATIO	15.30000	17.80000	17.80000	18.70000	18.70000
B	396.90000	396.90000	392.83000	394.63000	396.90000
LSTAT	4.98000	9.14000	4.03000	2.94000	5.33000
MEDV	24.00000	21.60000	34.70000	33.40000	36.20000

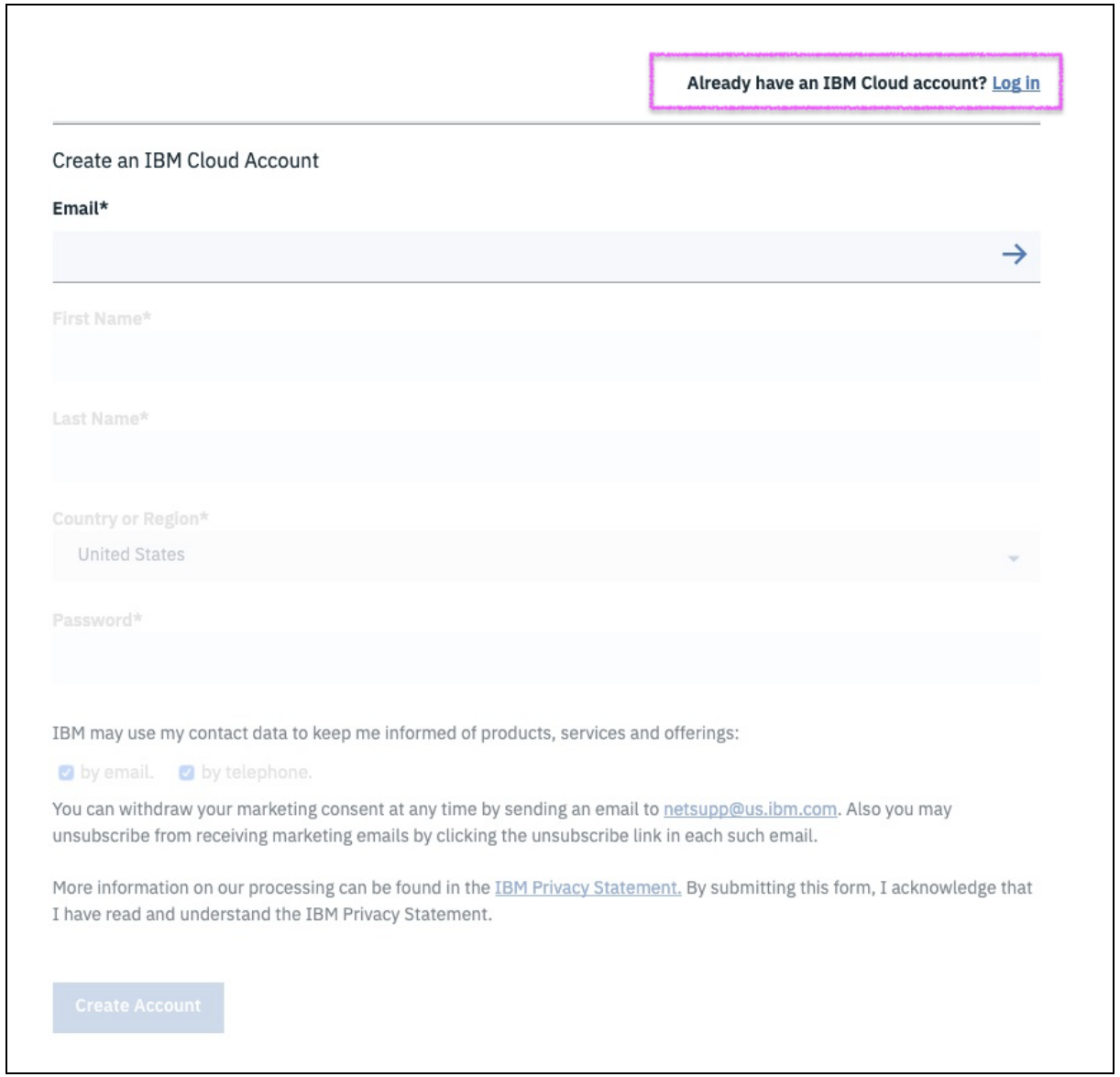
Linear regression - Simple



Steps

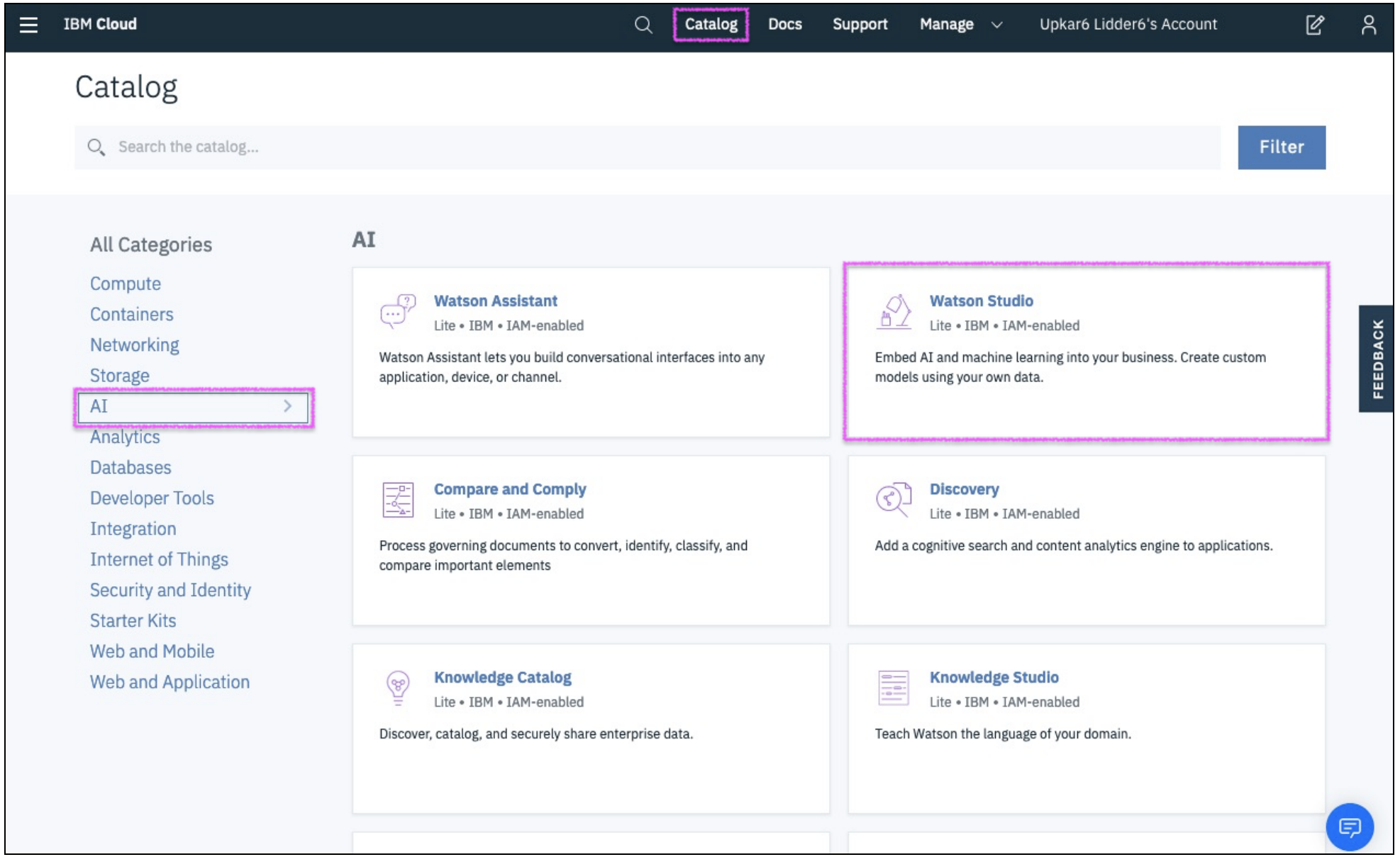
1. Sign up / Log into *IBM Cloud* - http://bit.ly/waston-ml-sign
2. Create *Watson Studio Service*.
3. Sign into Watson Studio and create a new *Data Science Project*. It also creates a *Cloud Object Store* for you.
4. Associate a *Machine Learning Service* with your project.
5. Upload csv *data* to your project.
6. Add a new *Machine Learning Model* to your project.
7. Create a *Linear Regression Model* and *save* it to IBM Cloud.
8. Create a new *deployment* on IBM Cloud.
9. *Test* your model !

Step 1 - sign up/ log into IBM Cloud

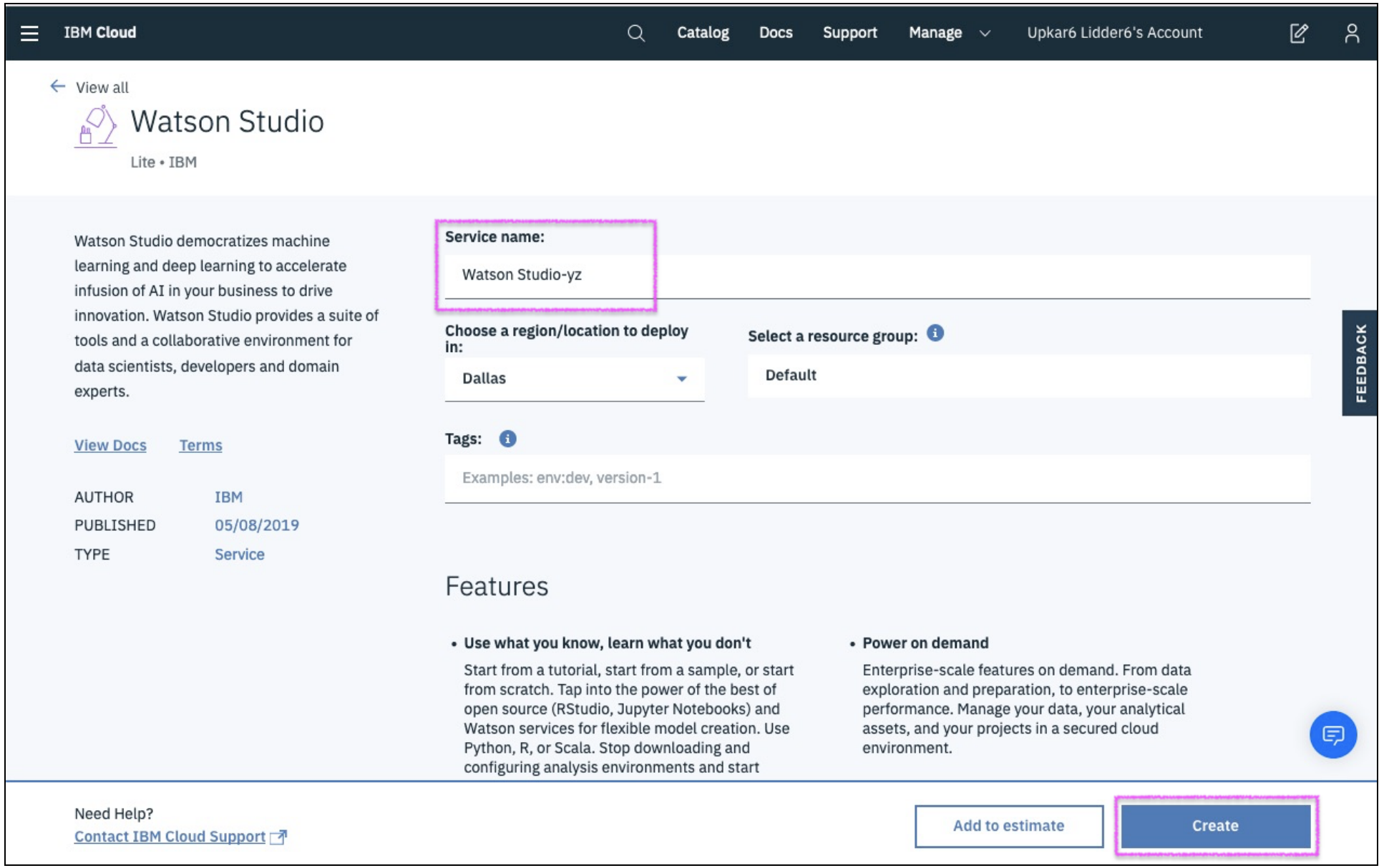


http://bit.ly/waston-ml-sign

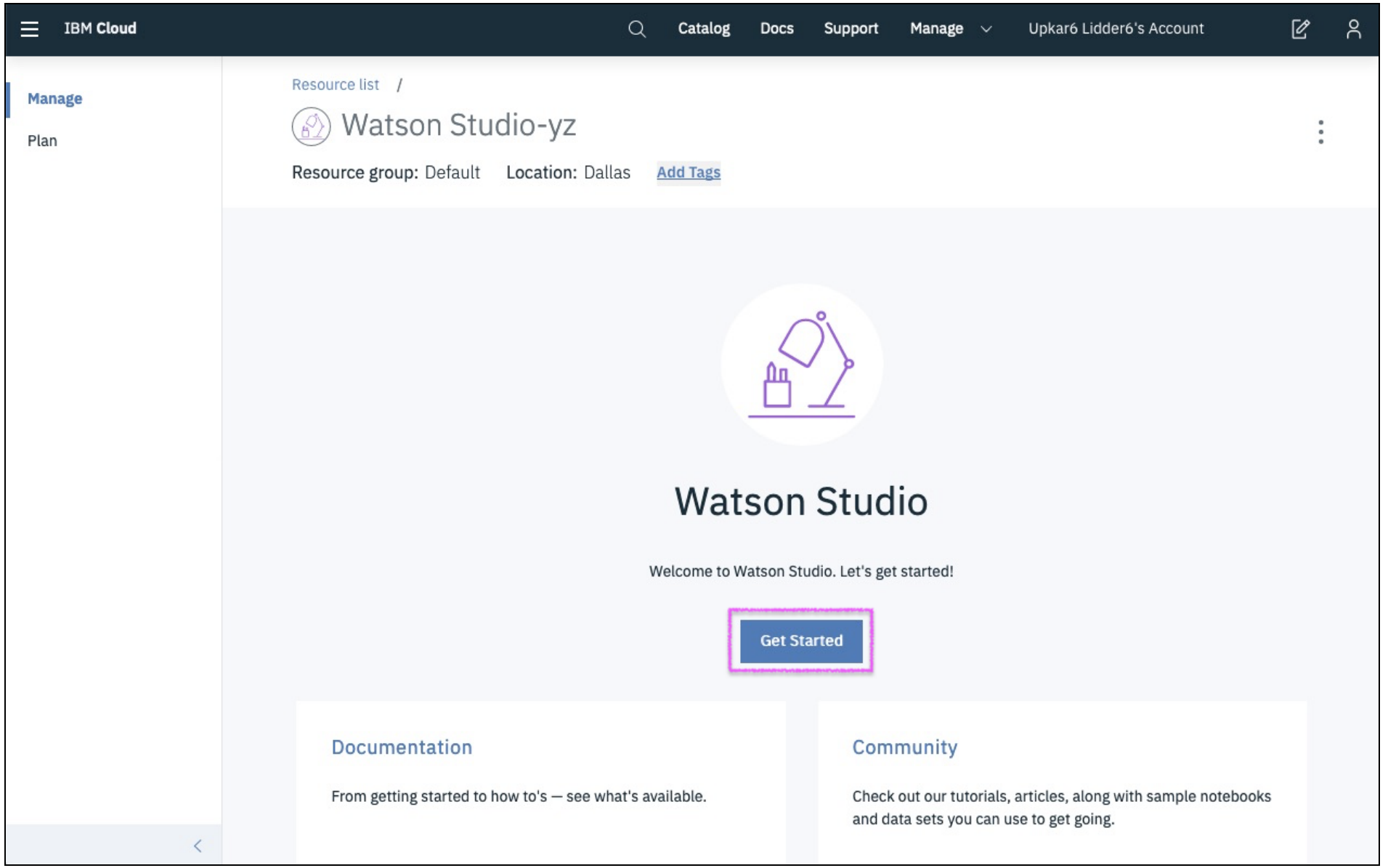
Step 2 - locate Watson Studio in Catalog



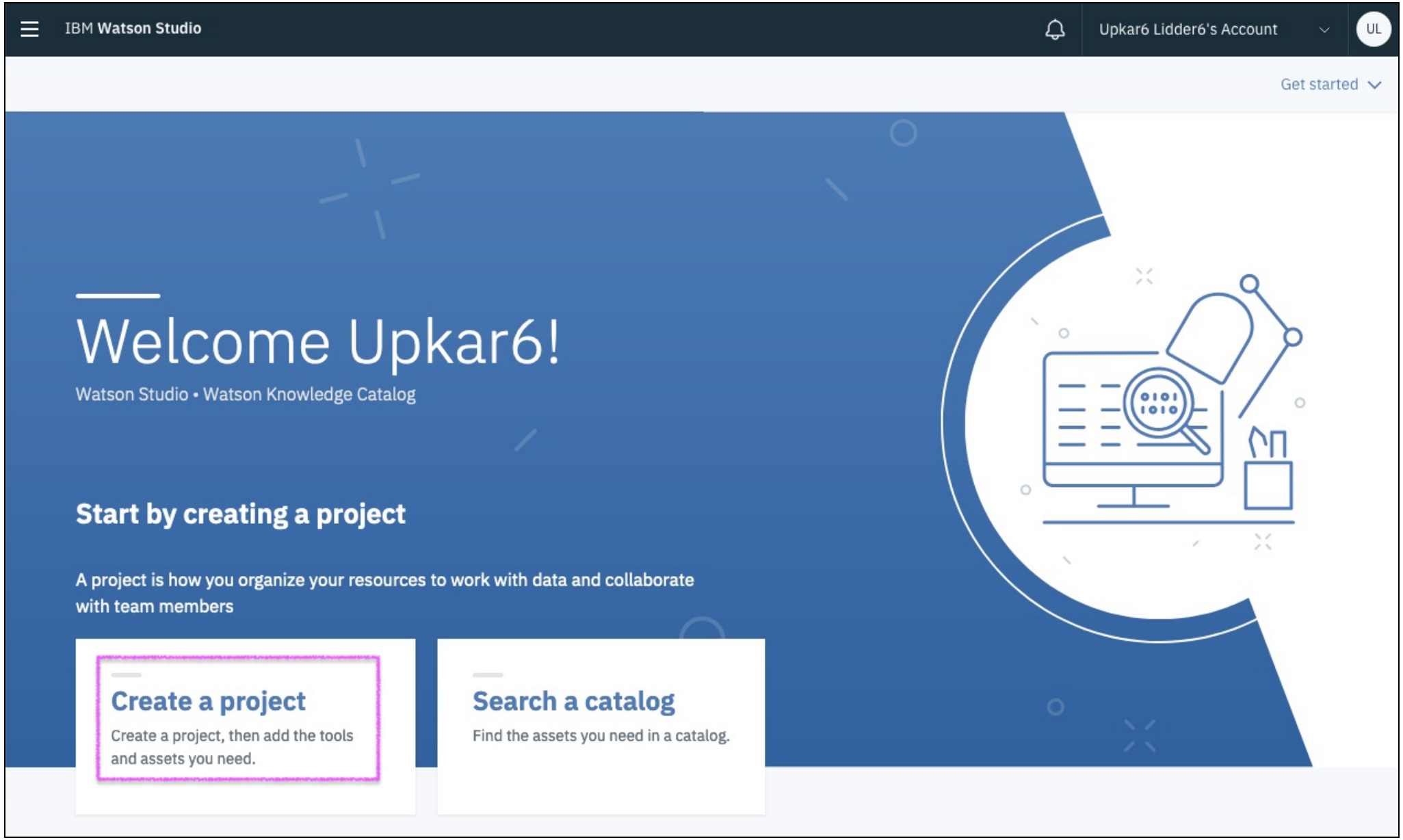
Step 3 - create Watson Studio instance



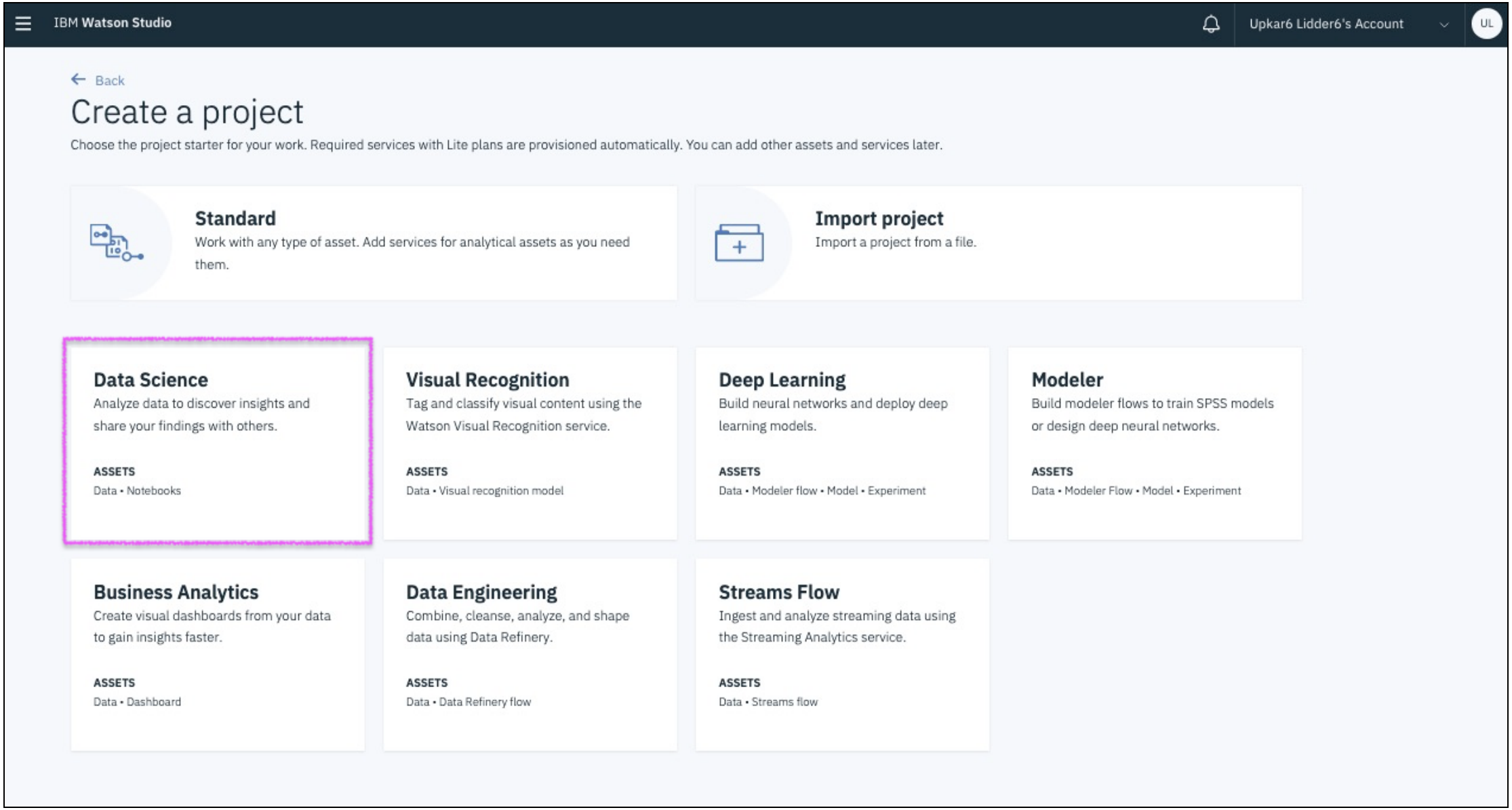
Step 4 - launch Watson Studio



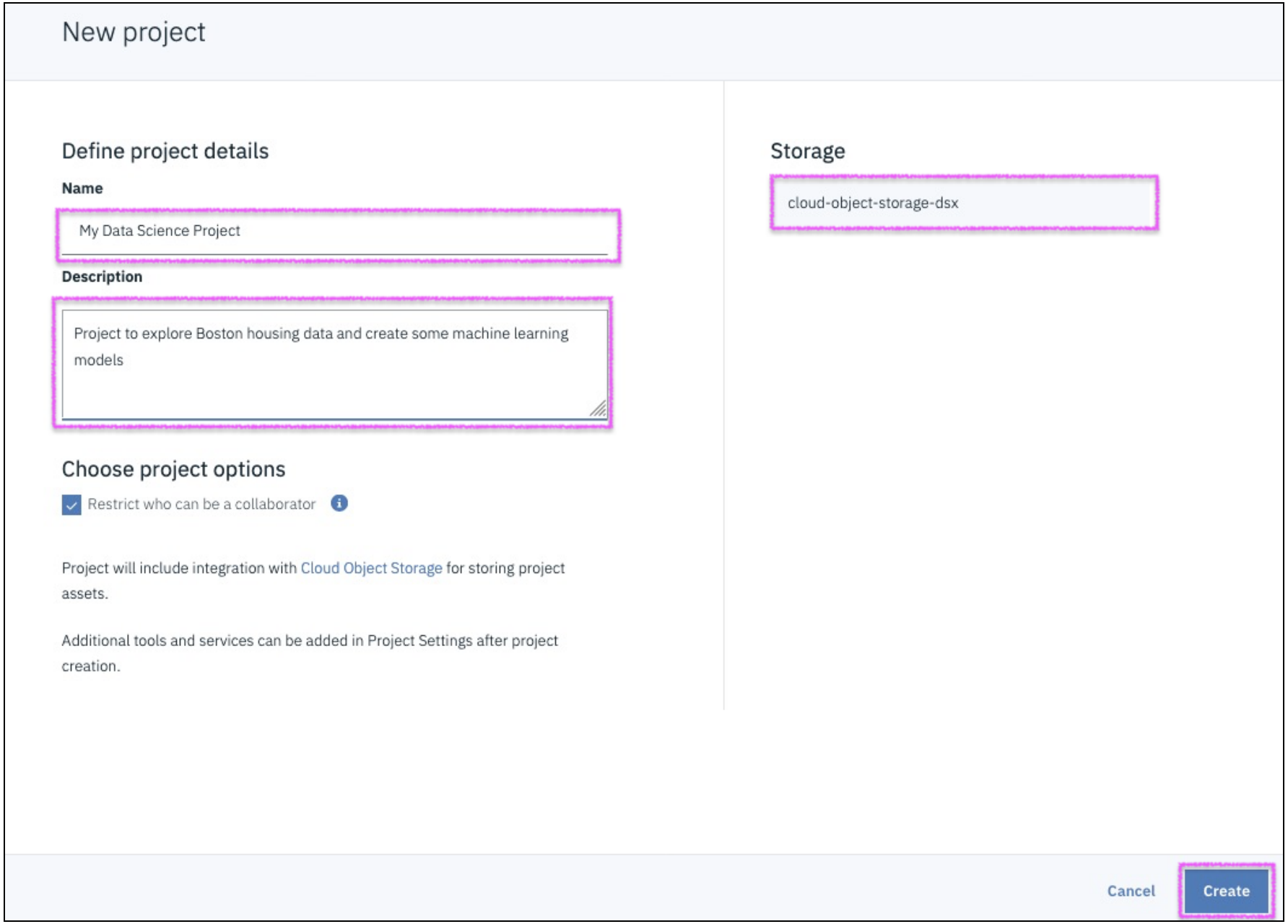
Step 5 - create a new project



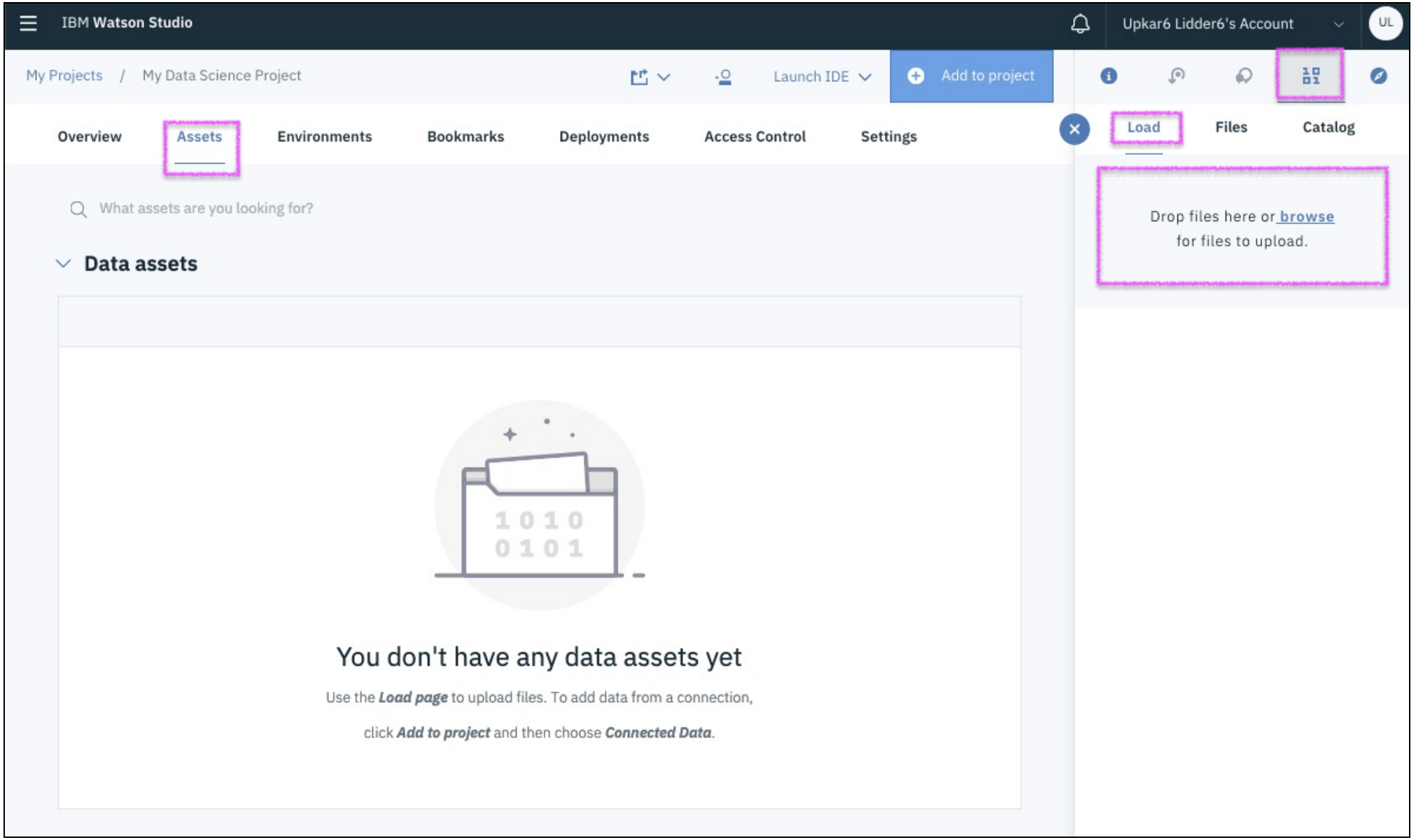
Step 6 - pick Data Science starter



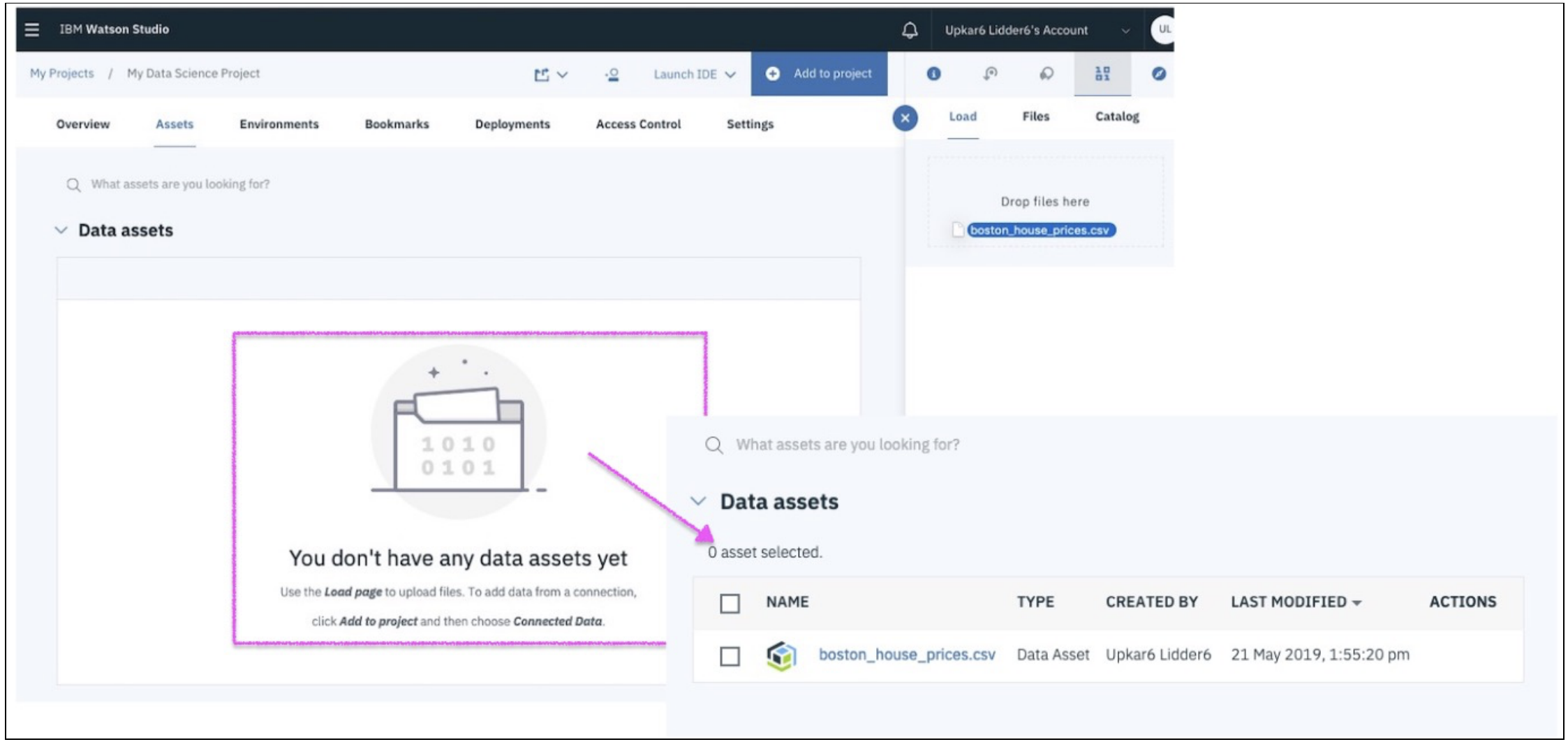
Step 7 - give the project a name and assign COS



Step 8 - open asset tab

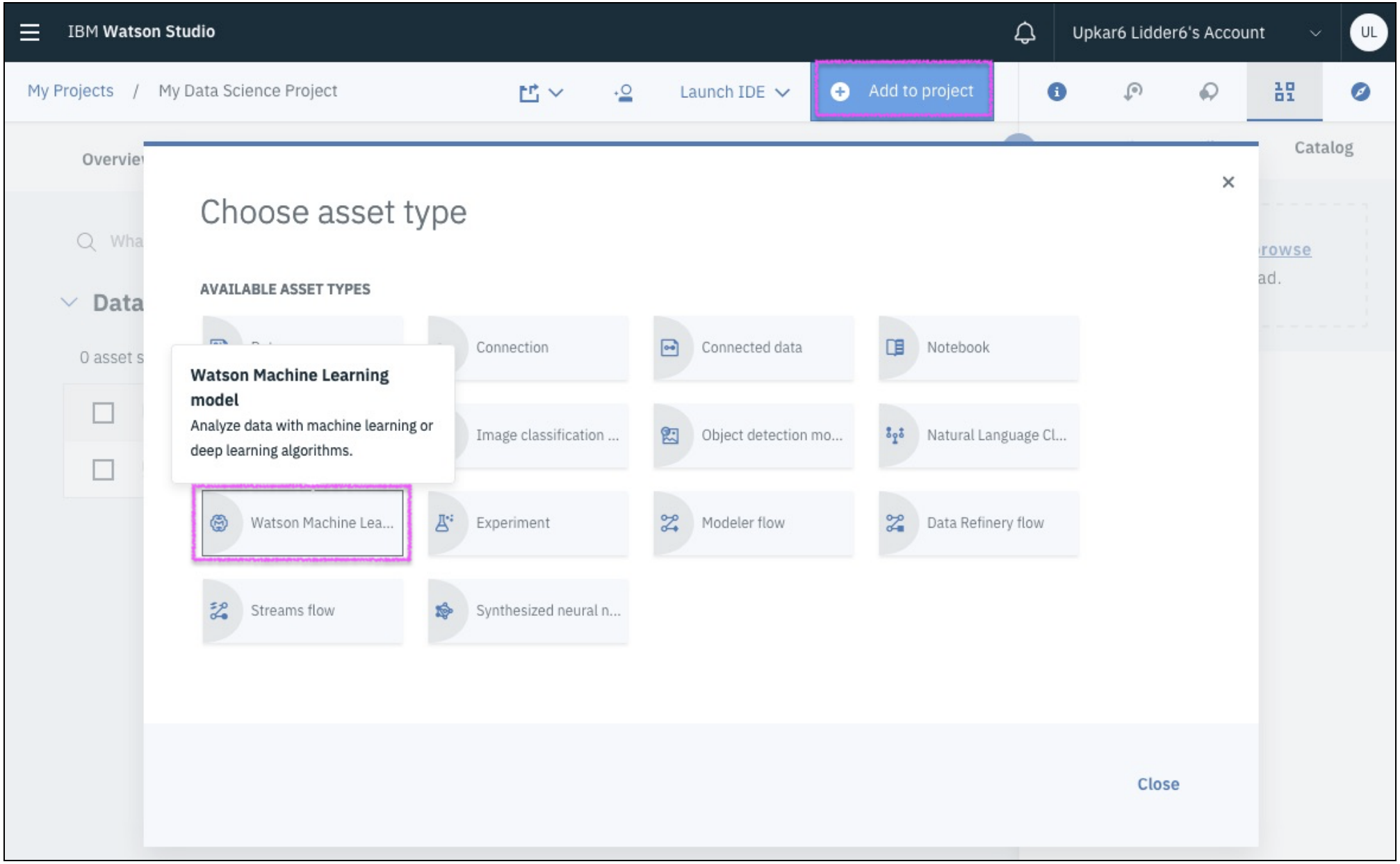


Step 9 - drag and drop data file into Load Assets

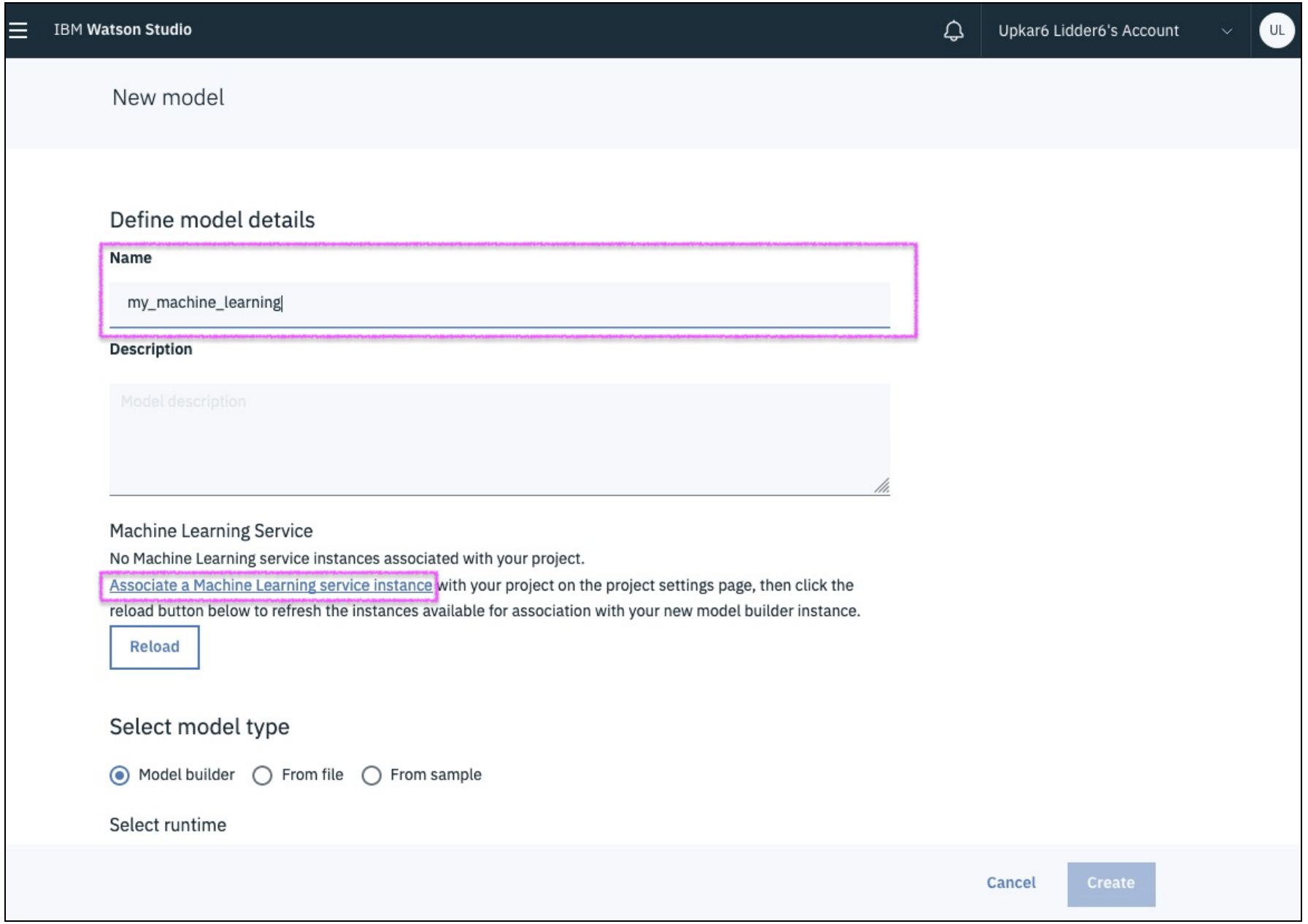


http://bit.ly/boston-house-csv

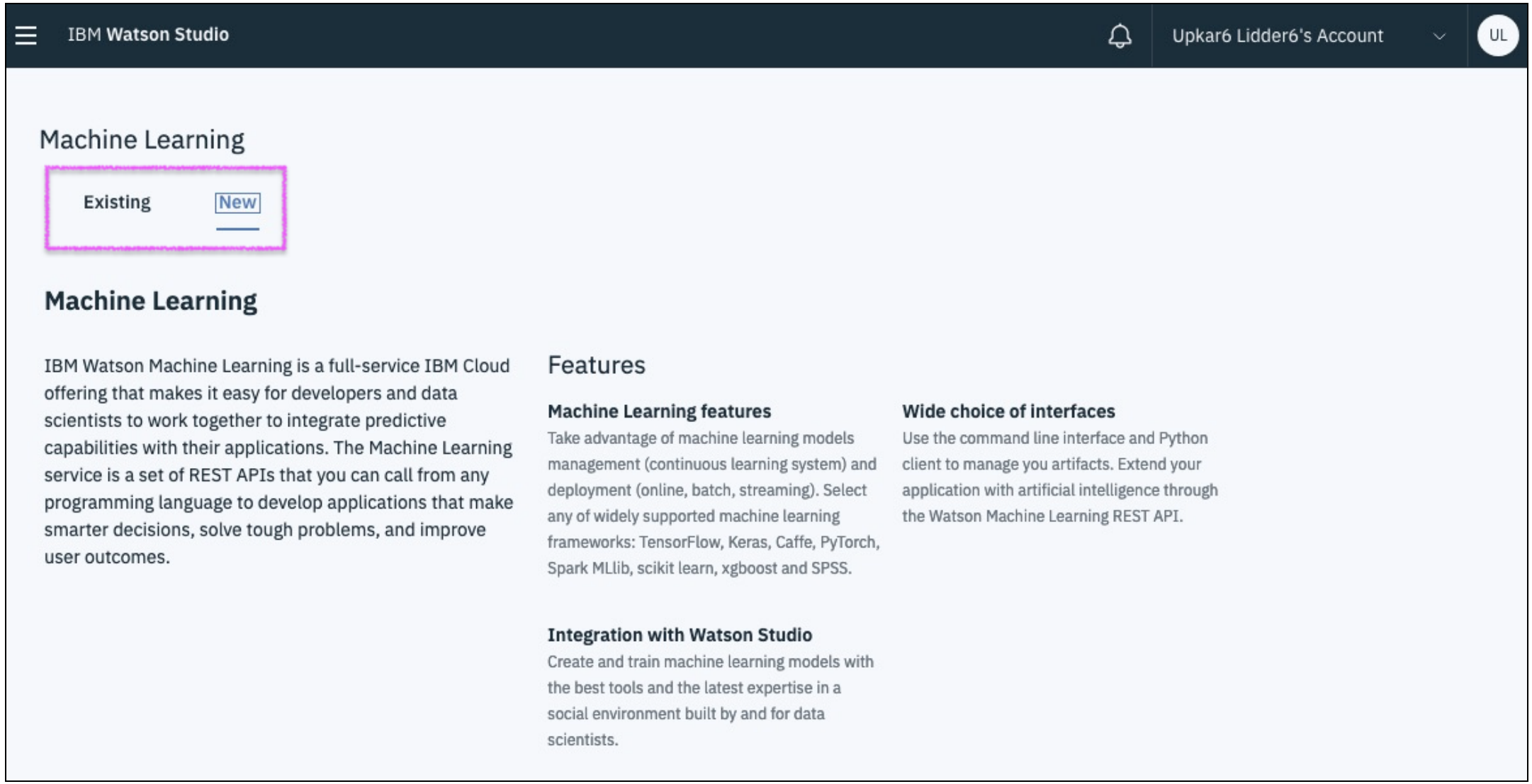
Step 10 - add Machine Learning model to the project



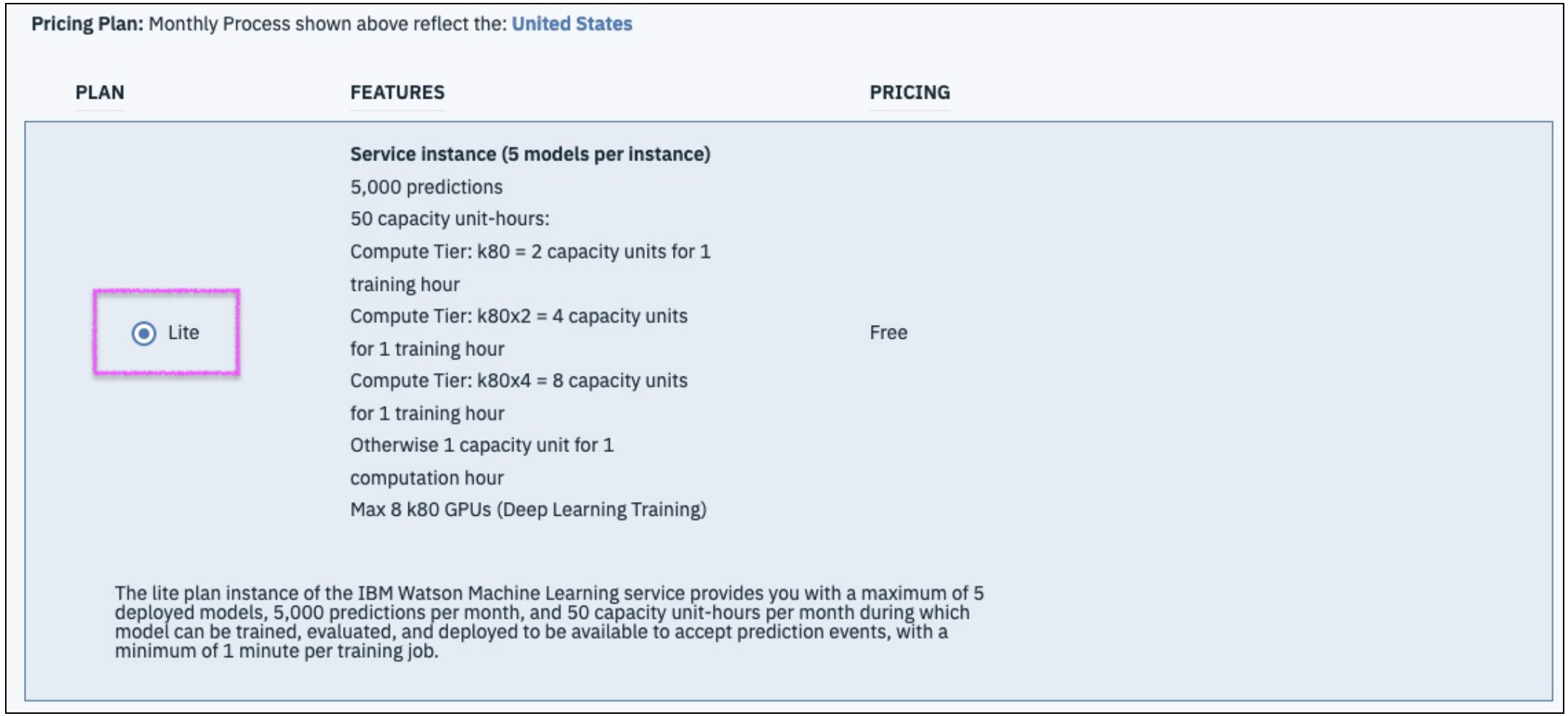
Step 11 - associate a ML instance



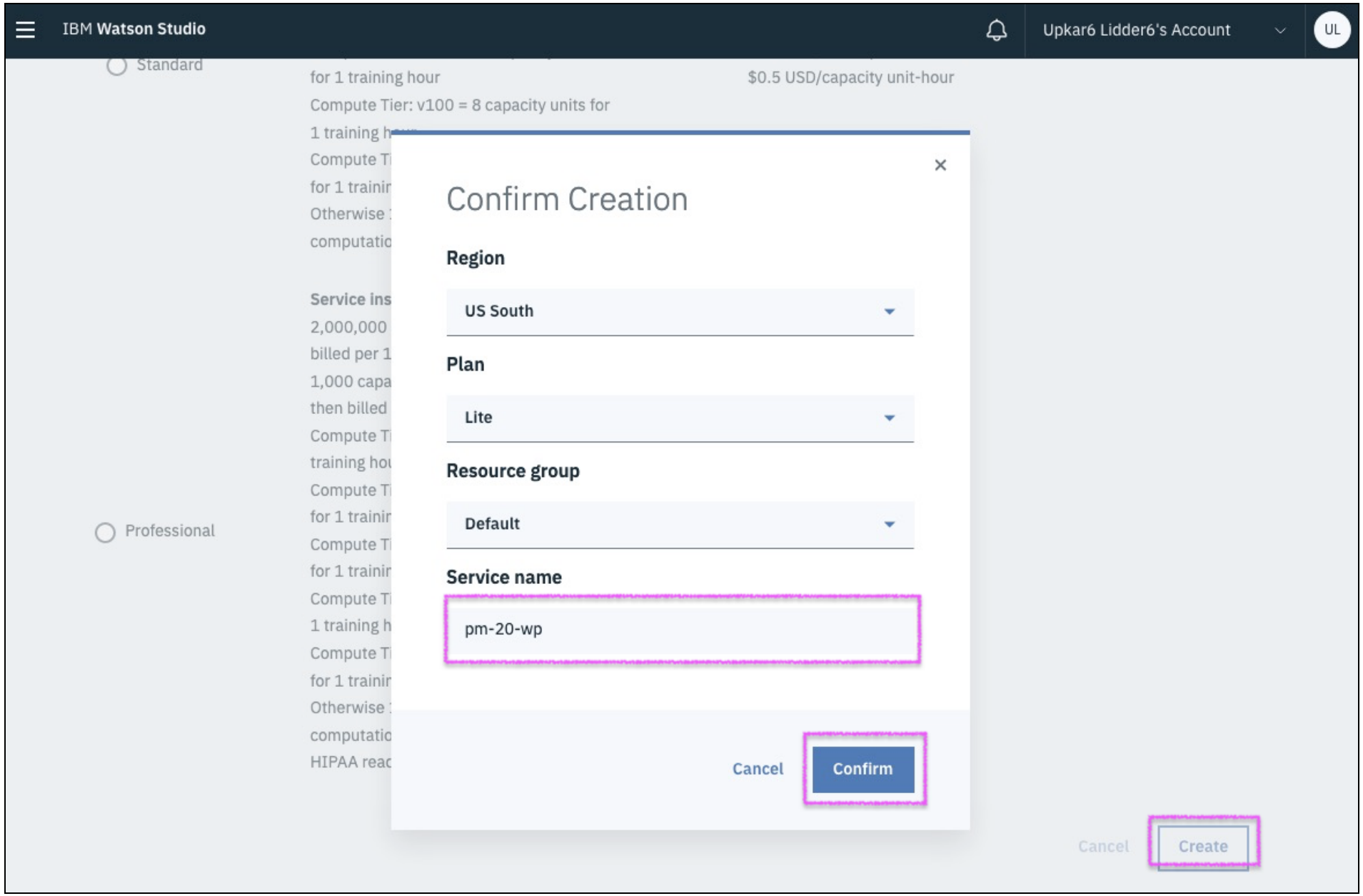
Step 12a - create ML instance if you don't have one



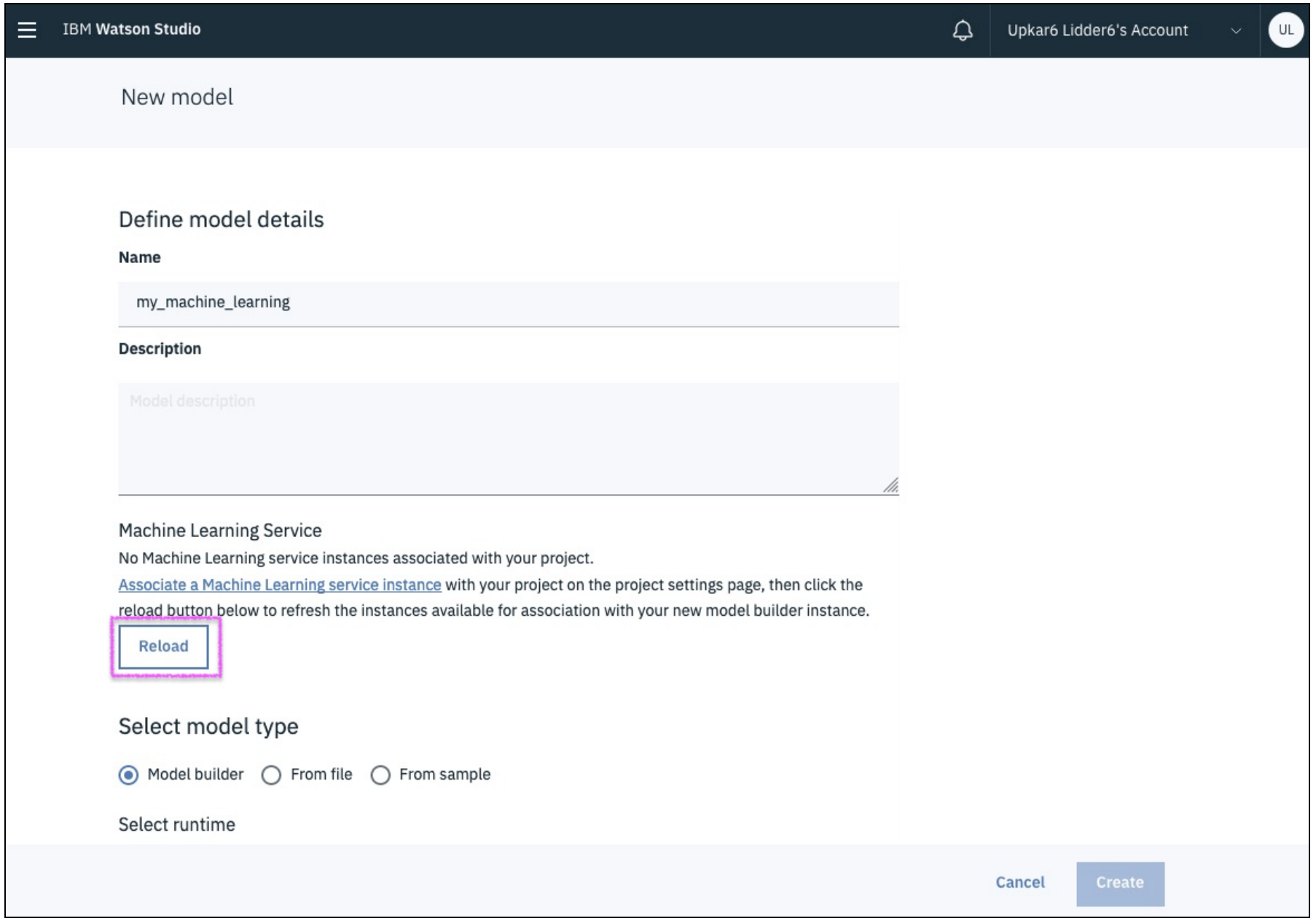
Step 12b - create a new lite ML instance



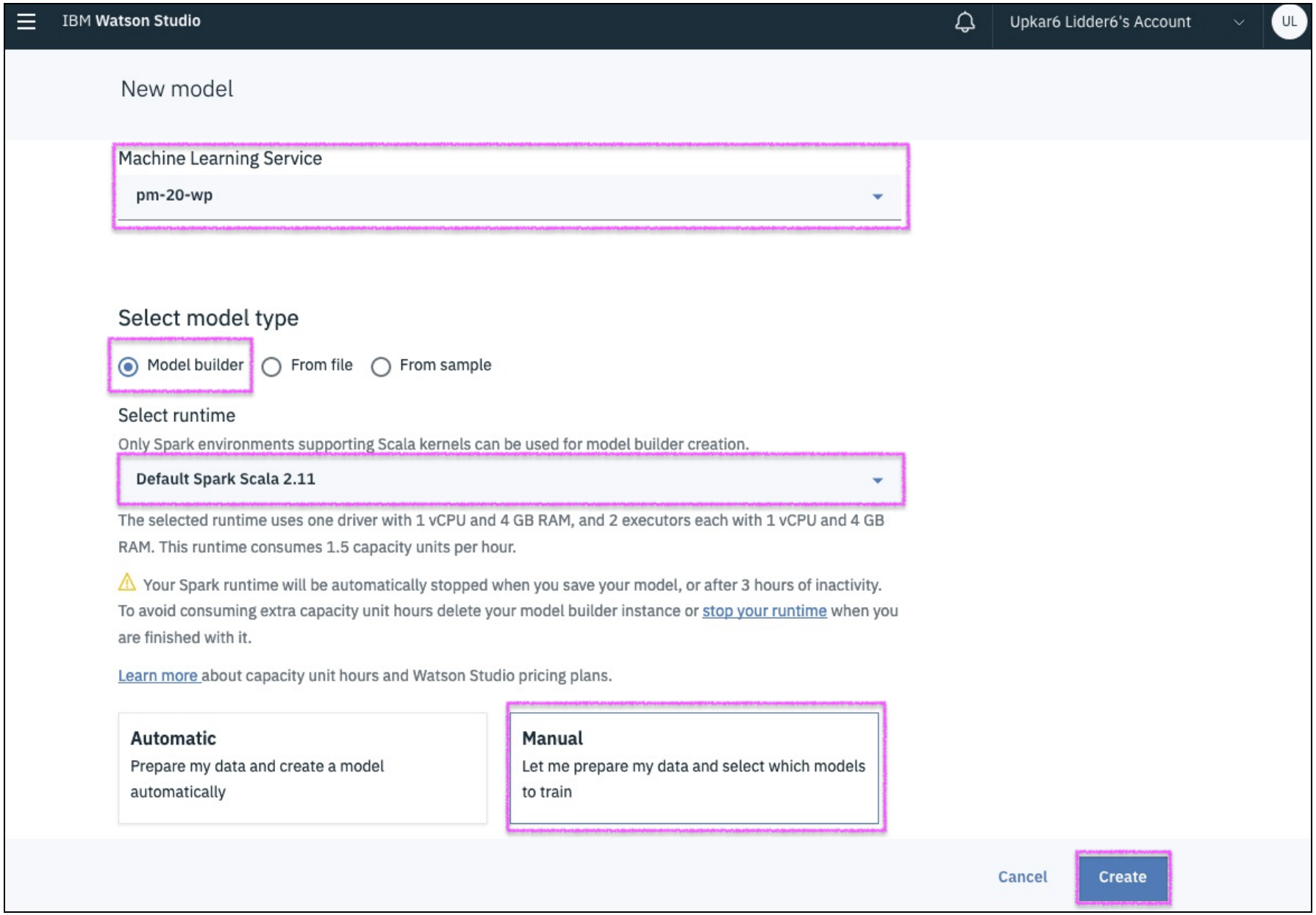
Step 12c - create a new lite ML instance



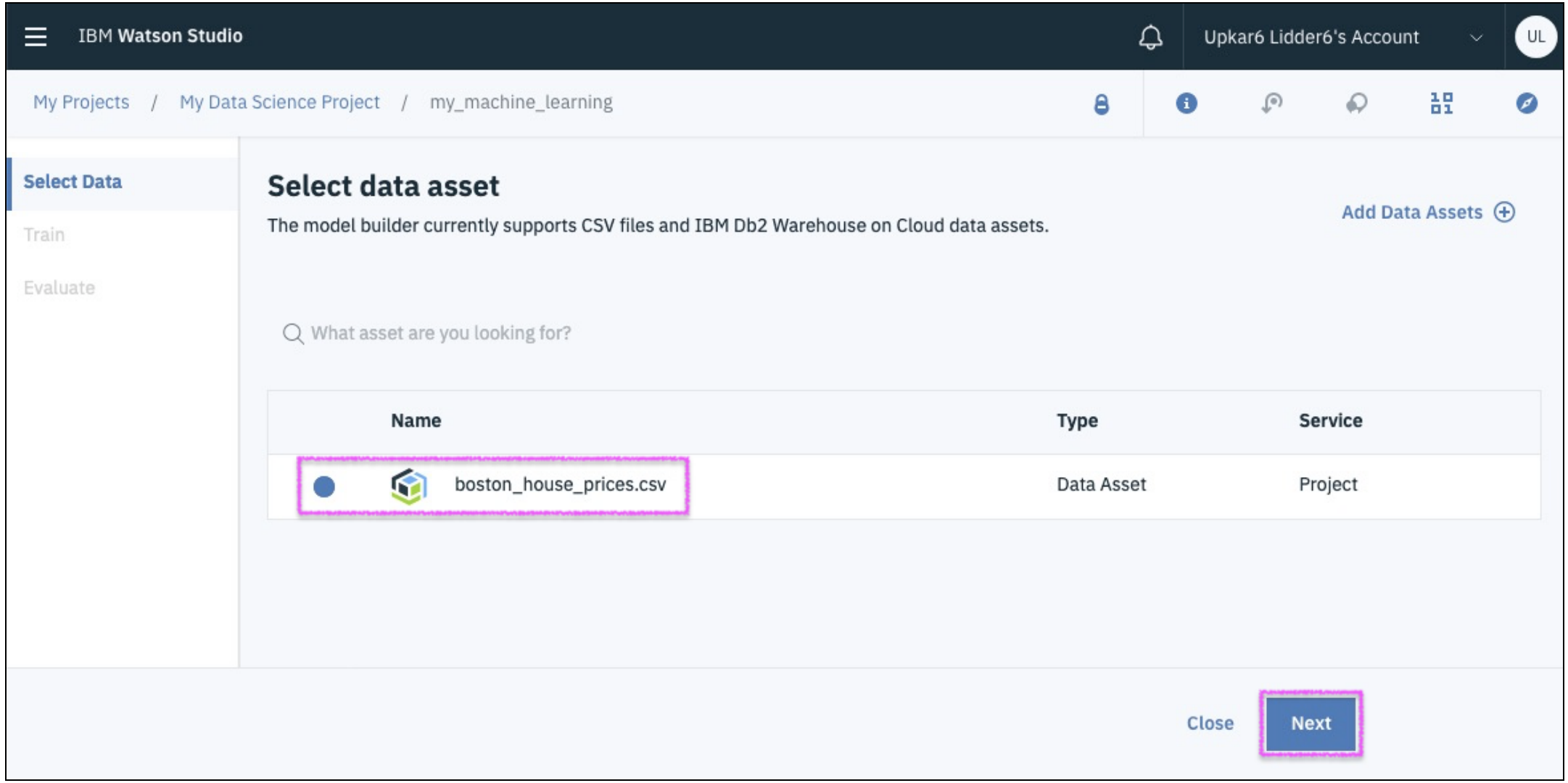
Step 12d - reload the ML service section



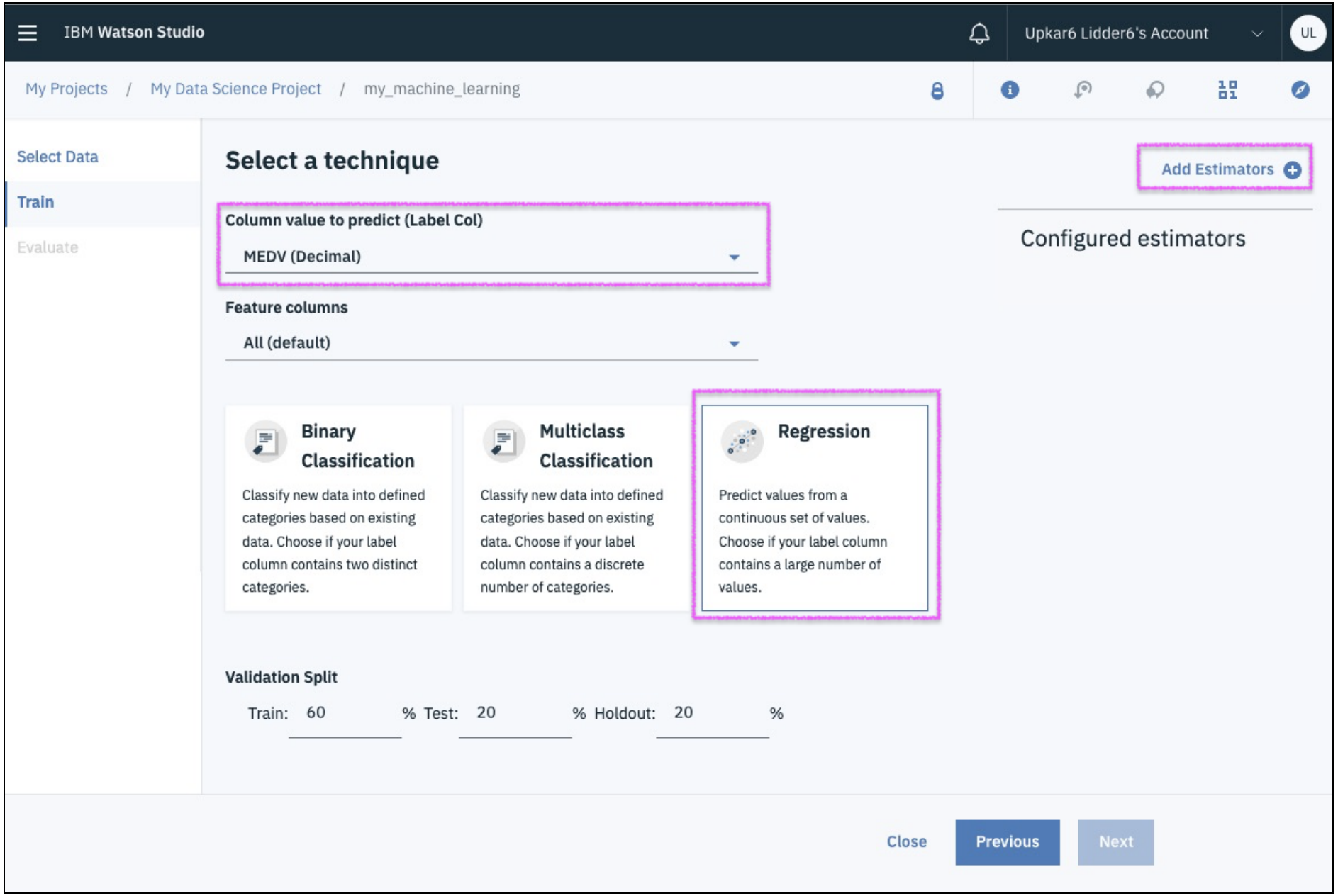
Step 13 - pick new service and create a model manually



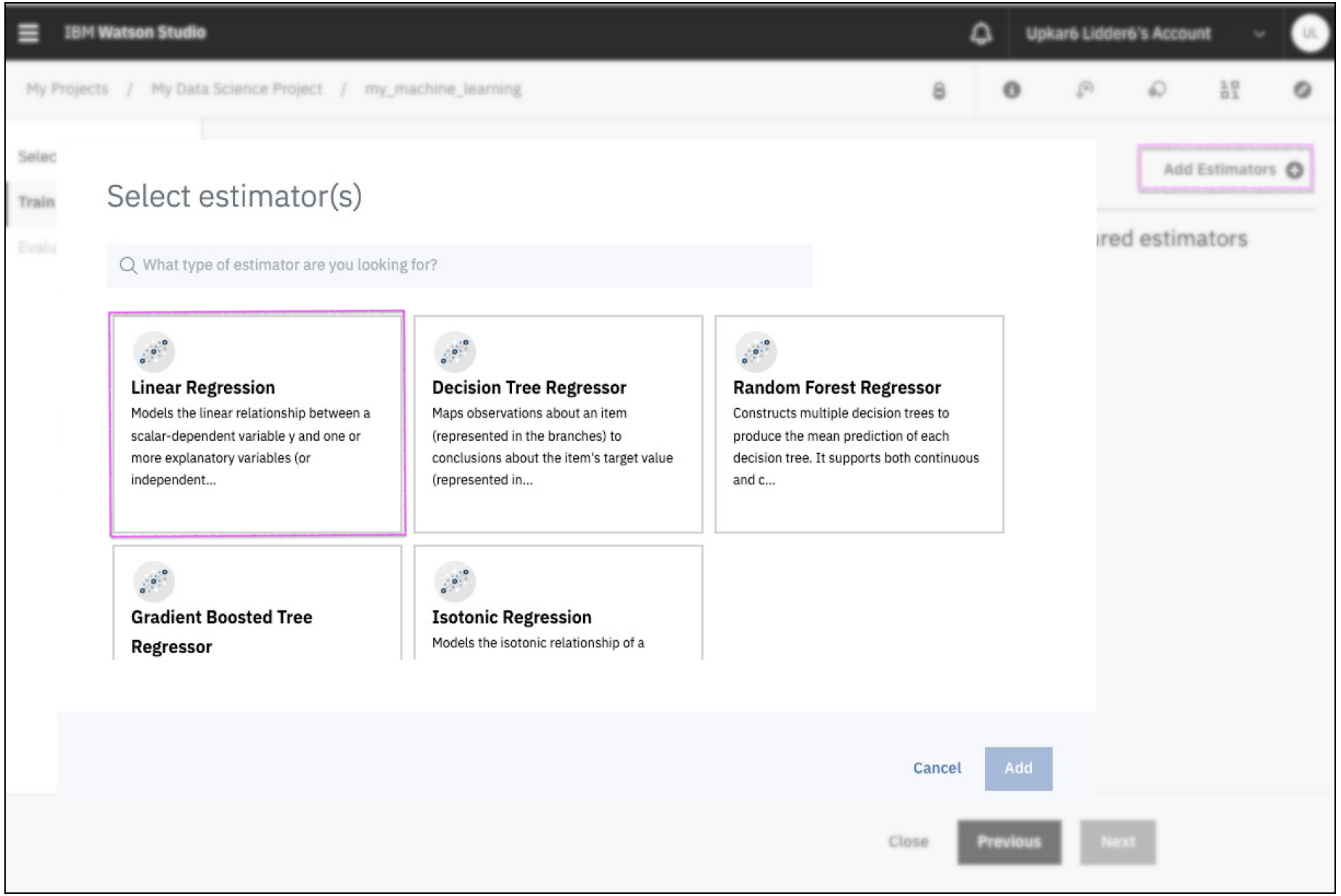
Step 14 - pick the Boston data file as *data asset*



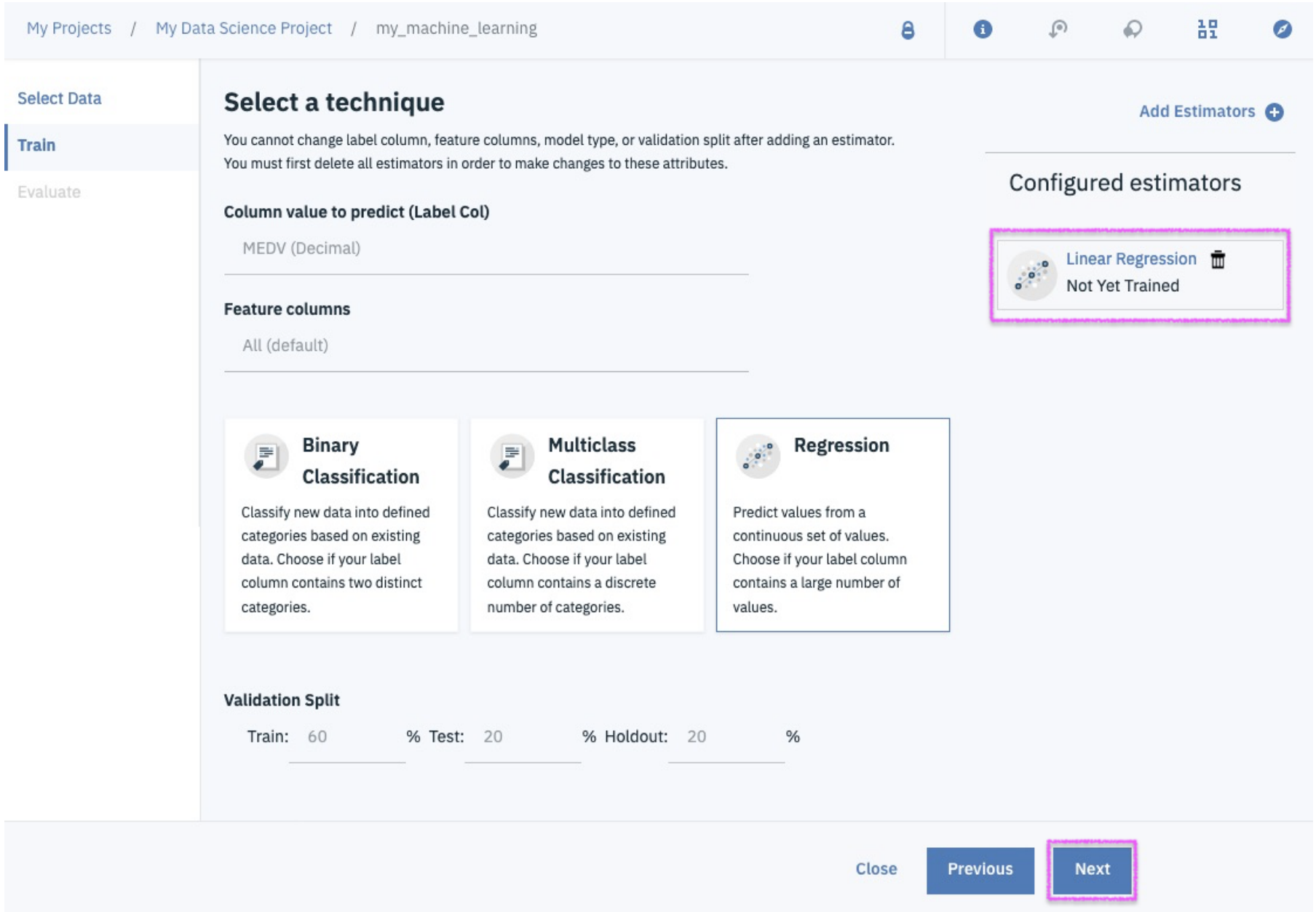
Step 15 - pick target, input features and model type



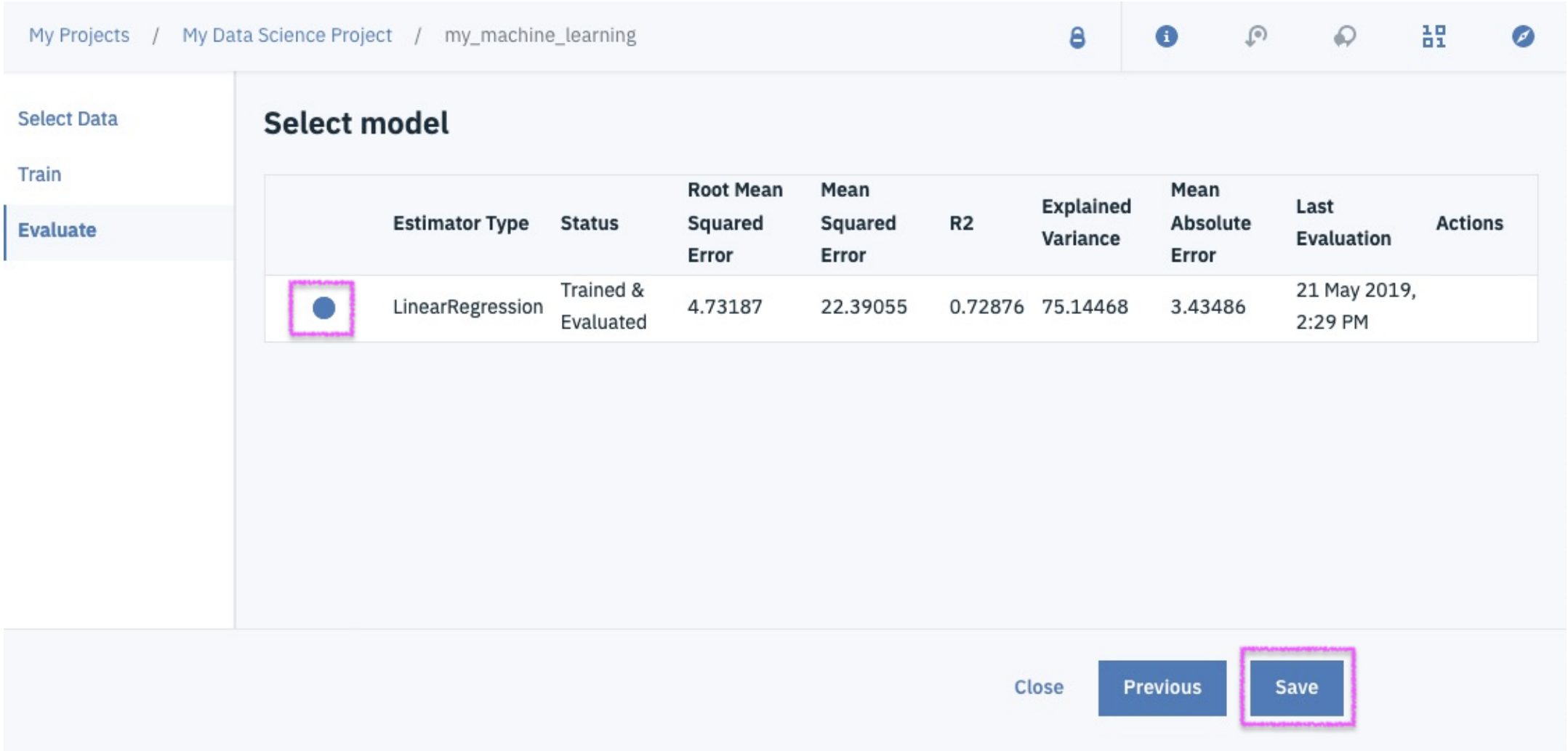
Step 16 - add estimator(s)



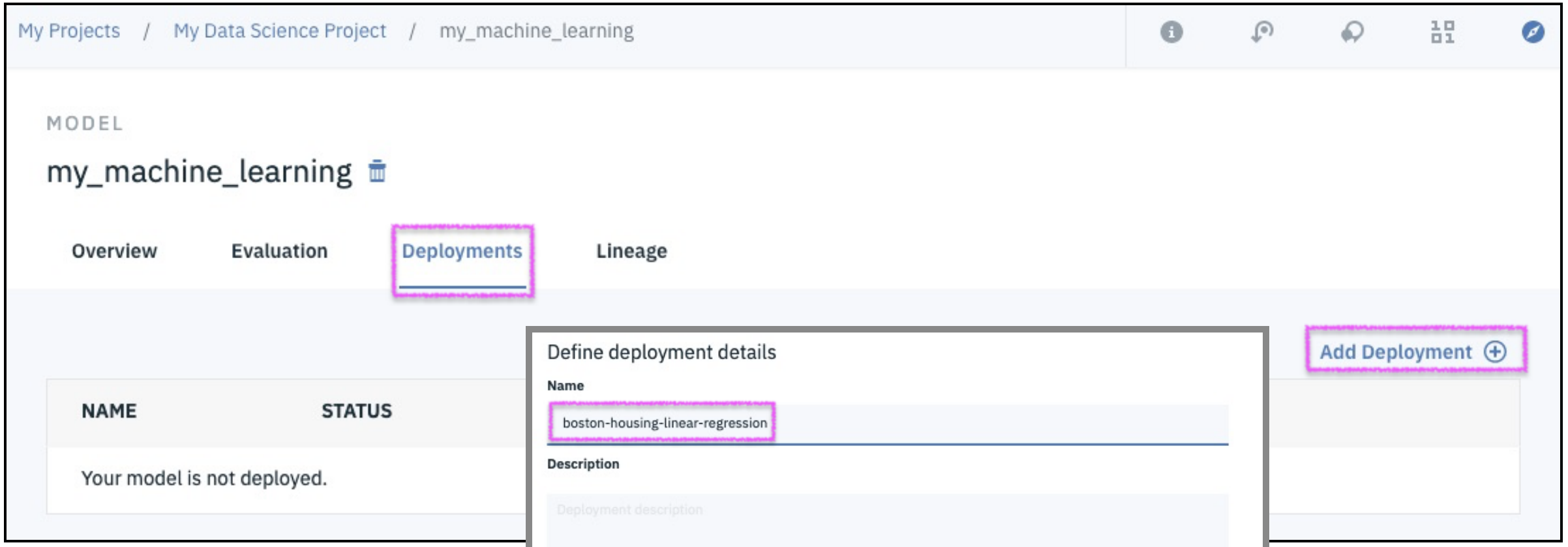
Step 17 - change hyperparameters (optional)



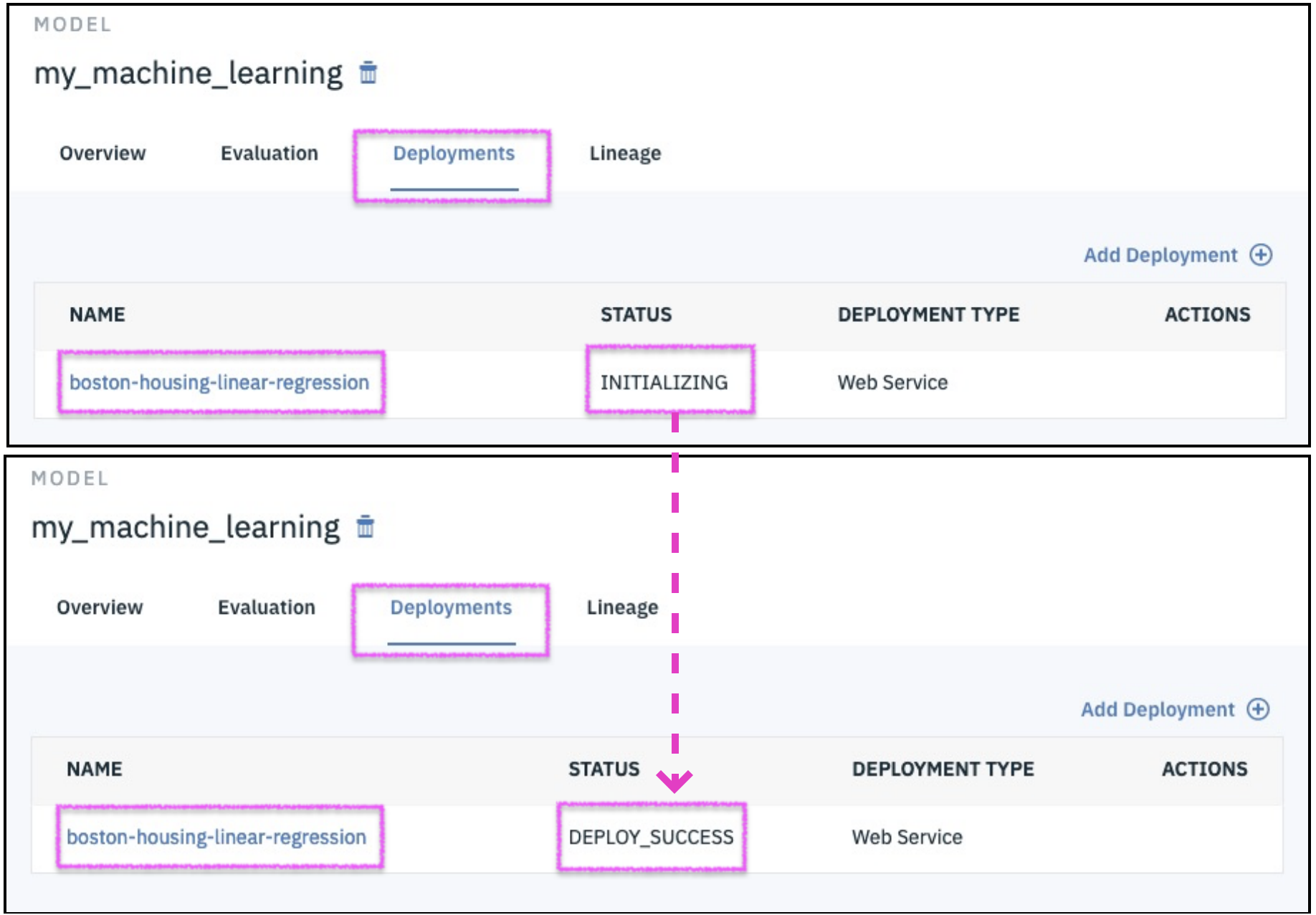
Step 18 - save/store the trained model on IBM Cloud



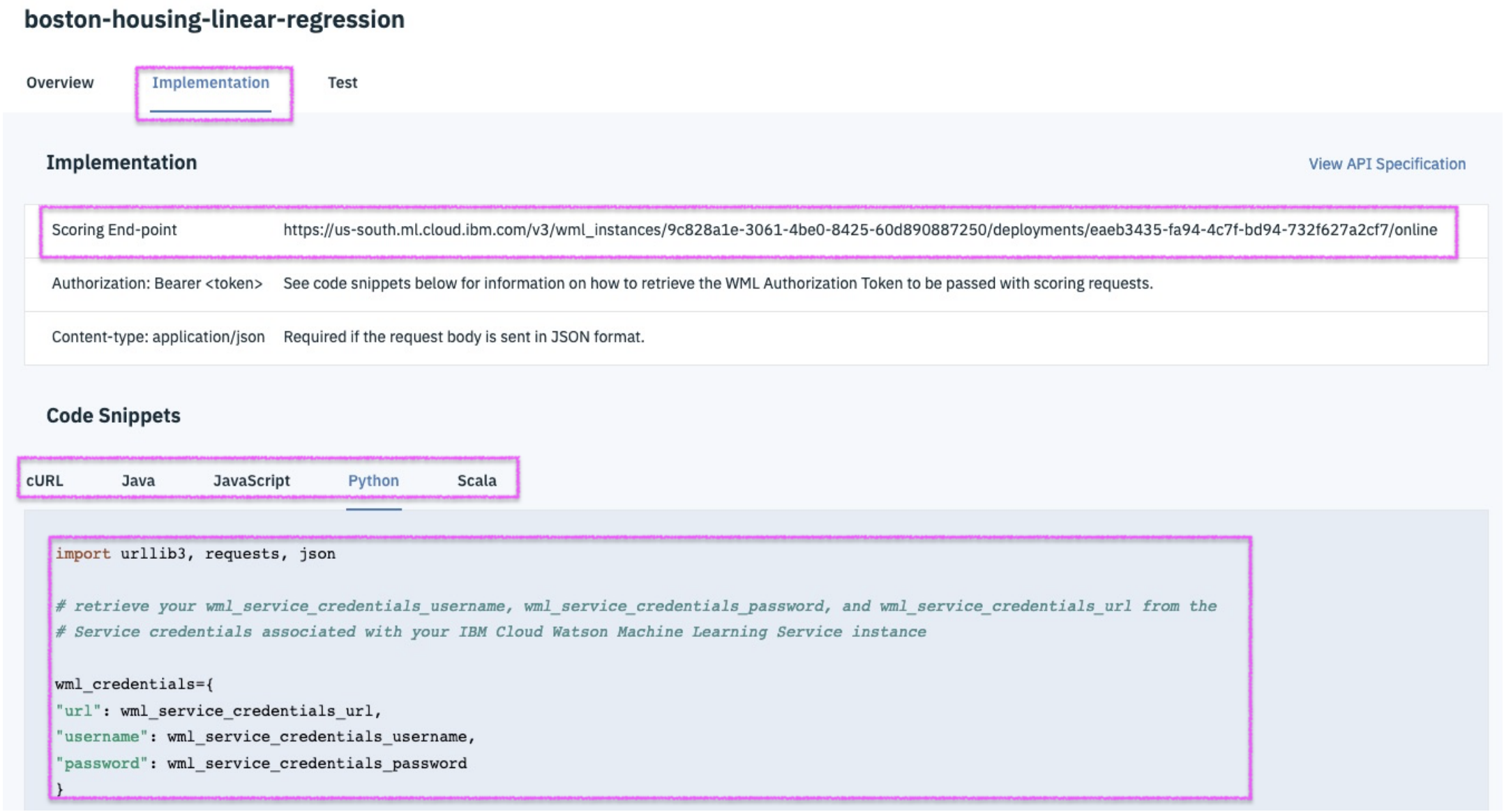
Step 19 - add a new deployment



Step 20 - ensure that deployment is successful

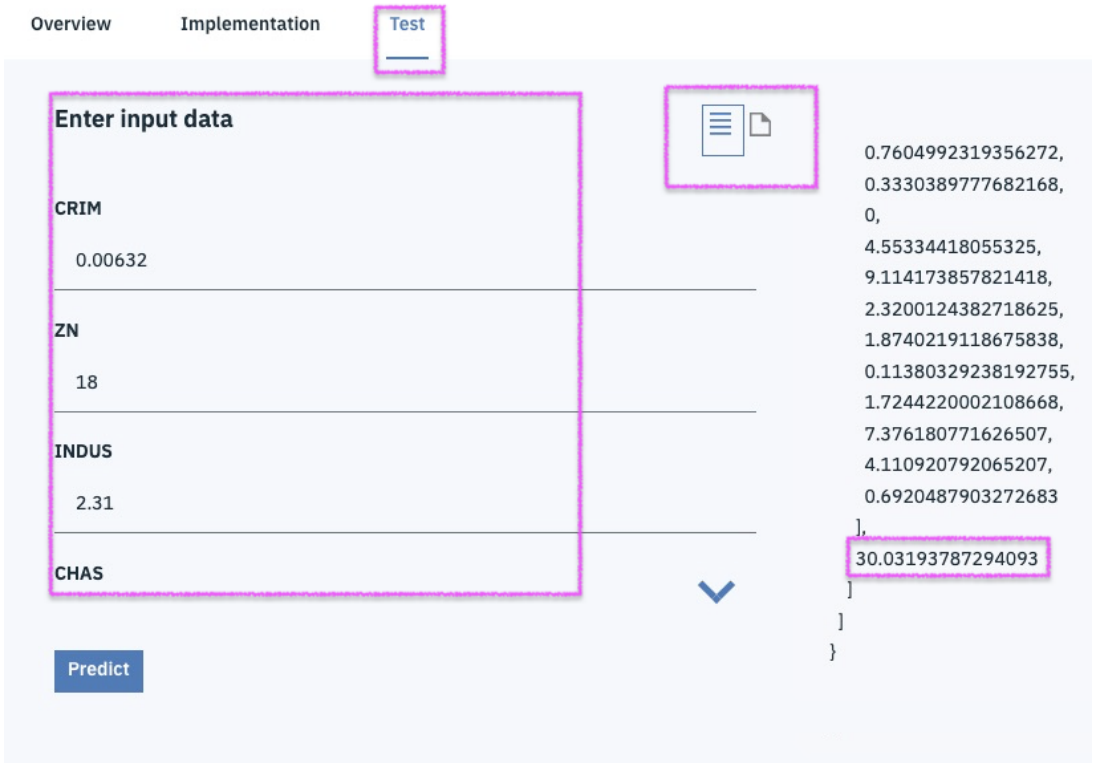


Step 21a - implementation / test the deployed model



Step 21b - implementation / test the deployed model

```
{
  "fields": ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"],
  "values": [
    [0.00632, 18, 2.31, 0, 0.538, 6.575, 66, 4.09, 1, 296, 15.3, 396.9, 4.99]
  ]
}
```



Watson Studio - Binary Classification

Estimator	Description
Logistic regression	Analyzes a data set in which there are one or more independent variables that determine one of two outcomes. Only binary logistic regression is supported
Decision tree classifier	Maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It supports both binary and multiclass labels, as well as both continuous and categorical features.
Random forest classifier	Constructs multiple decision trees to produce the label that is a mode of each decision tree. It supports both binary and multiclass labels, as well as both continuous and categorical features.
Gradient boosted tree classifier	Produces a classification prediction model in the form of an ensemble of decision trees. It only supports binary labels, as well as both continuous and categorical features.

Watson Studio - Multiclass Classification

Estimator	Description
Decision tree classifier	Maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It supports both binary and multiclass labels, as well as both continuous and categorical features.
Random forest classifier	Constructs multiple decision trees to produce the label that is a mode of each decision tree. It supports both binary and multiclass labels, as well as both continuous and categorical features.
Naive Bayes	Classifies features based on Bayes' theorem, which assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Watson Studio - Regression

Estimator	Description
Linear regression	Models the linear relationship between a scalar-dependent variable y and one or more explanatory variables (or independent variables) x.
Decision tree regressor	Maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It supports both continuous and categorical features.
Random forest regressor	Constructs multiple decision trees to produce the mean prediction of each decision tree. It supports both continuous and categorical features.
Gradient boosted tree regressor	Produces a regression prediction model in the form of an ensemble of decision trees. It supports both continuous and categorical features.
Isotonic regression	Models the isotonic relationship of a sequence of observations by fitting a free-form line to the observations under the following constraints: the fitted free-form line must be non-decreasing everywhere, and it must lie as close to the observations as possible.

Watson Machine Learning

WML - Supported Frameworks as of 06.21.19

TensorFlow

- Version 1.5
- Version 1.11 in an Anaconda 5.2.0 environment

Spark MLlib

Spark 2.1

Caffe

Version 1.0

Predictive Model Markup Language (PMML)

Version 3.0 to 4.3

XGBoost

- XGBoost 0.6a2 and 0.71 in an Anaconda 4.2.x environment with scikit-learn 0.17 and Python 3.5
- XGBoost 0.80 in an Anaconda 5.0.1 environment with scikit-learn 0.19 and Python 3.5

scikit-learn

- scikit-learn 0.17 on Anaconda 4.2.x for Python 3.5 Runtime
- scikit-learn 0.19 on Anaconda 5.0.0 for Python 3.5 Runtime

PyTorch

Versions: 0.3, 0.4.1, 1.0

Keras

- Version 2.1.3 with Tensorflow version 1.5
- Version 2.2.4 with TensorFlow version 1.11 in an Anaconda 5.2.0 environment

IBM SPSS Modeler

- IBM SPSS Modeler 17.1
- IBM SPSS Modeler 18.0

IBM Watson Machine Learning

Embed Machine Learning and Deep Learning in your Business

Push analytics to Data

ML close to data for faster and secure insights

- Connect to remote Hadoop and Spark clusters to train ML models at scale
- Move analytics to the data, access data from Hadoop and combine it with RDBMS to expand data access and optimize performance.
- Simplify big data analysis for analysts and business users.
- Mix and match deployment options

Deploy & Manage

Manage models and versions

- Save models on a central repository with version control built in
- Portable models – deploy in the cloud, on devices or on premise
- Import models trained somewhere else and deploy in WML
- Transfer models to connected devices (e.g. Core ML, Tensorflow Lite)
- Deployment flexibility: Shiny apps, scripts, decision optimization and WEX models

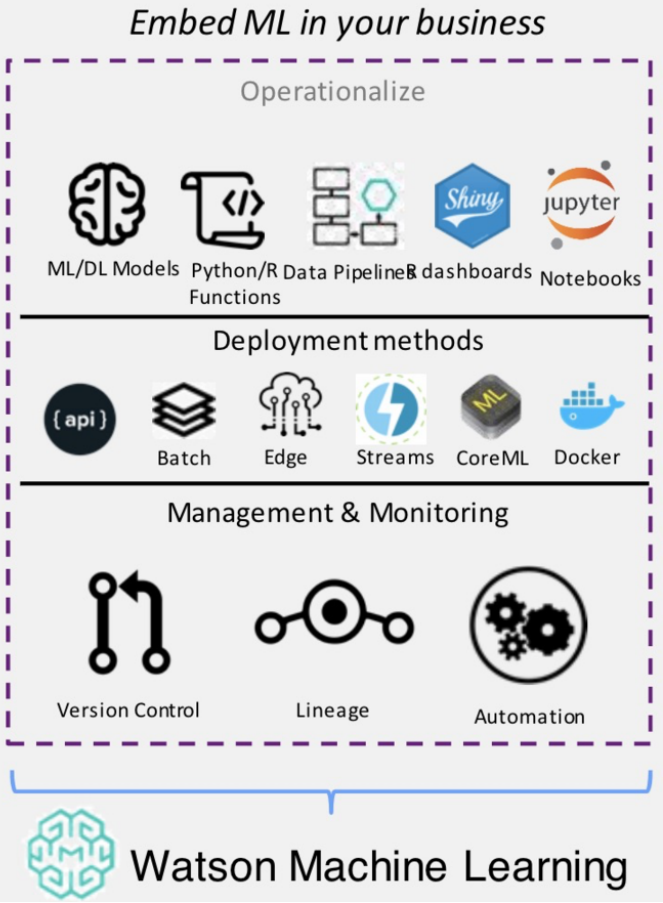
Automate ML

Your models always learning with a feedback loop

- Automate the retraining of models with feedback loop capabilities
- Automate the deployment of models in products
- Automate the Hyperparameter Optimization and Feature Engineering to rapidly train models
- A/B testing of models and performance Monitoring

Fully Managed on IBM Cloud

Deploy it on Private or Public Cloud



IBM Watson Machine Learning Client

Welcome to watson-machine-learning-client's documentation!

`watson-machine-learning-client` is a python library that allows to work with Watson Machine Learning service on IBM Cloud. Test and deploy your models as APIs for application development, share with colleagues using this python library.

Contents

http://wml-api-pyclient.mybluemix.net/index.html

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - create scikit-learn linear regression model

```
1  from sklearn.linear_model import LinearRegression
2  from sklearn.datasets import load_boston
3  import matplotlib.pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error, r2_score
6
7  boston = load_boston()
8  X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predited data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

Output

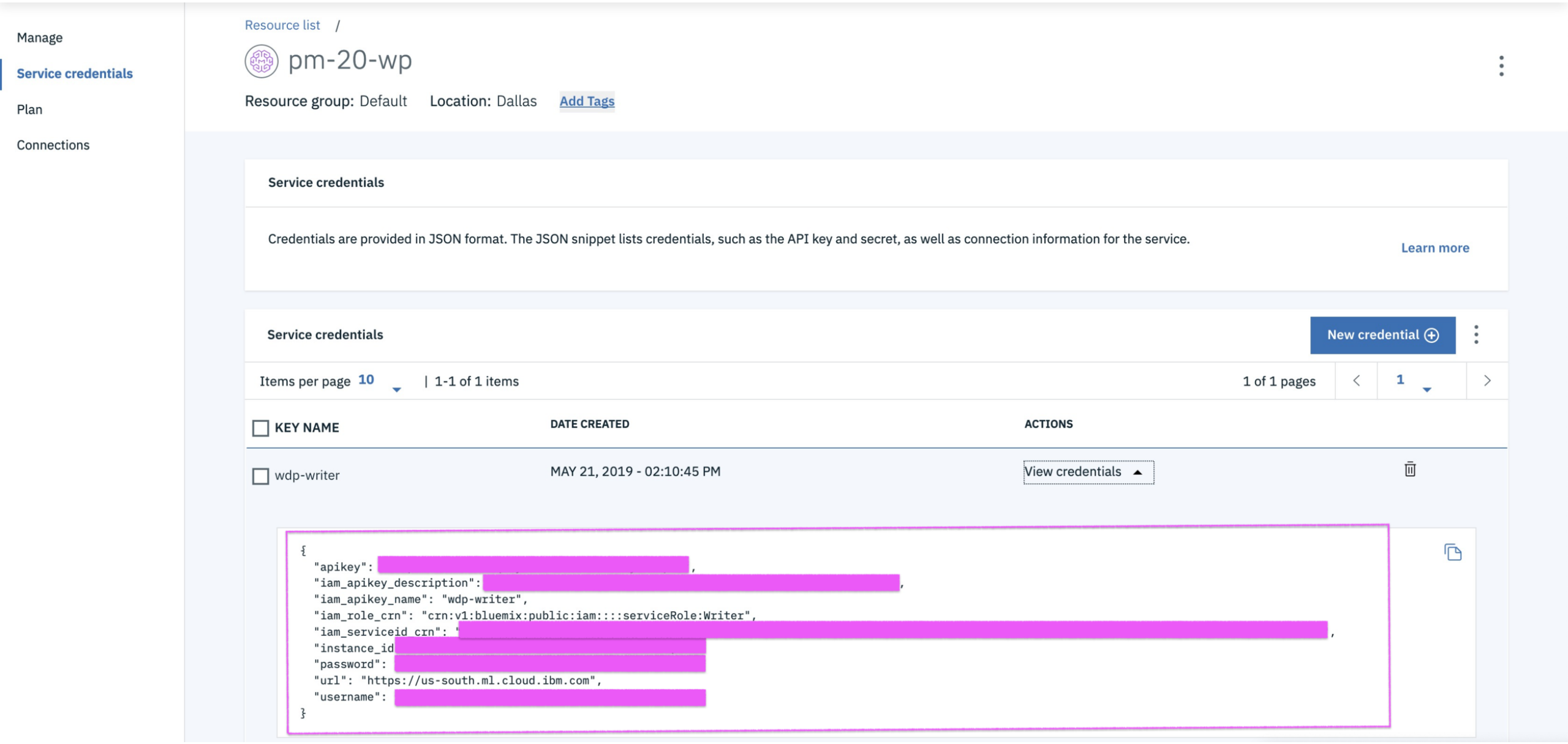
```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - evaluation metrics

Version	a38f373b-44cd-4a1e-9cb5-87343a69da12
Phase	setup
meanSquaredError	19.775
rootMeanSquaredError	4.447
r2	0.751
meanAbsoluteError	3.223
explainedVariance	74.316

```
Coefficients:
 [ -1.05001380e-01   3.29973196e-02   3.09915520e-02   2.74529793e+00
  -2.00482943e+01   4.60880933e+00  -1.36509230e-03  -1.27392294e+00
   2.69375701e-01  -1.12477385e-02  -9.48125356e-01   8.76288422e-03
  -4.24918821e-01]
Mean squared error: 33.77
Variance score: 0.67
```

WML - get Machine Learning service credentials



WML - save scikit-learn linear regression model

```
# we will use WML to work with IBM Machine Learning Service
from watson_machine_learning_client import WatsonMachineLearningAPIClient

# Grab your credentials from the Watson Service section in Watson Studio or IBM Cloud Dashboard
wml_credentials = {
}

# Instantiate WatsonMachineLearningAPIClient
from watson_machine_learning_client import WatsonMachineLearningAPIClient
client = WatsonMachineLearningAPIClient( wml_credentials )

# store the model
published_model = client.repository.store_model(model=LR_model,
                                                meta_props={'name':'upkar-housing-linear-reg'},
                                                training_data=X_train, training_target=y_train)
```

WML - save scikit-learn linear regression model

```
# we will use WML to work with IBM Machine Learning Service
from watson_machine_learning_client import WatsonMachineLearningAPIClient

# Grab your credentials from the Watson Service section in Watson Studio or IBM Cloud Dashboard
wml_credentials = {
}

# Instantiate WatsonMachineLearningAPIClient
from watson_machine_learning_client import WatsonMachineLearningAPIClient
client = WatsonMachineLearningAPIClient( wml_credentials )

# store the model
published_model = client.repository.store_model(model=LR_model,
                                                meta_props={'name':'upkar-housing-linear-reg'},
                                                training_data=X_train, training_target=y_train)
```

WML - save scikit-learn linear regression model

```
# we will use WML to work with IBM Machine Learning Service
from watson_machine_learning_client import WatsonMachineLearningAPIClient

# Grab your credentials from the Watson Service section in Watson Studio or IBM Cloud Dashboard
wml_credentials = {
}

# Instantiate WatsonMachineLearningAPIClient
from watson_machine_learning_client import WatsonMachineLearningAPIClient
client = WatsonMachineLearningAPIClient( wml_credentials )

# store the model
published_model = client.repository.store_model(model=LR_model,
                                                meta_props={'name':'upkar-housing-linear-reg'},
                                                training_data=X_train, training_target=y_train)
```

WML - save scikit-learn linear regression model

```
# we will use WML to work with IBM Machine Learning Service
from watson_machine_learning_client import WatsonMachineLearningAPIClient

# Grab your credentials from the Watson Service section in Watson Studio or IBM Cloud Dashboard
wml_credentials = {
}

# Instantiate WatsonMachineLearningAPIClient
from watson_machine_learning_client import WatsonMachineLearningAPIClient
client = WatsonMachineLearningAPIClient( wml_credentials )

# store the model
published_model = client.repository.store_model(model=LR_model,
                                                meta_props={'name':'upkar-housing-linear-reg'},
                                                training_data=X_train, training_target=y_train)
```

WML - deploy scikit-learn linear regression model

```
import json

# grab the model from IBM Cloud
published_model_uid = client.repository.get_model_uid(published_model)

# create a new deployment for the model
model_deployed = client.deployments.create(published_model_uid, "Deployment of scikit model")

#get the scoring endpoint
scoring_endpoint = client.deployments.get_scoring_url(model_deployed)
print(scoring_endpoint)

#use the scoring endpoint to predict house median price some test data
scoring_payload = {"values": [list(X_test[0]), list(X_test[1])]}
predictions = client.deployments.score(scoring_endpoint, scoring_payload)
print(json.dumps(predictions, indent=2))
```

WML - deploy scikit-learn linear regression model

```
import json

# grab the model from IBM Cloud
published_model_uid = client.repository.get_model_uid(published_model)

# create a new deployment for the model
model_deployed = client.deployments.create(published_model_uid, "Deployment of scikit model")

#get the scoring endpoint
scoring_endpoint = client.deployments.get_scoring_url(model_deployed)
print(scoring_endpoint)

#use the scoring endpoint to predict house median price some test data
scoring_payload = {"values": [list(X_test[0]), list(X_test[1])]}
predictions = client.deployments.score(scoring_endpoint, scoring_payload)
print(json.dumps(predictions, indent=2))
```

WML - deploy scikit-learn linear regression model

```
import json

# grab the model from IBM Cloud
published_model_uid = client.repository.get_model_uid(published_model)

# create a new deployment for the model
model_deployed = client.deployments.create(published_model_uid, "Deployment of scikit model")

#get the scoring endpoint
scoring_endpoint = client.deployments.get_scoring_url(model_deployed)
print(scoring_endpoint)

#use the scoring endpoint to predict house median price some test data
scoring_payload = {"values": [list(X_test[0]), list(X_test[1])]}
predictions = client.deployments.score(scoring_endpoint, scoring_payload)
print(json.dumps(predictions, indent=2))
```

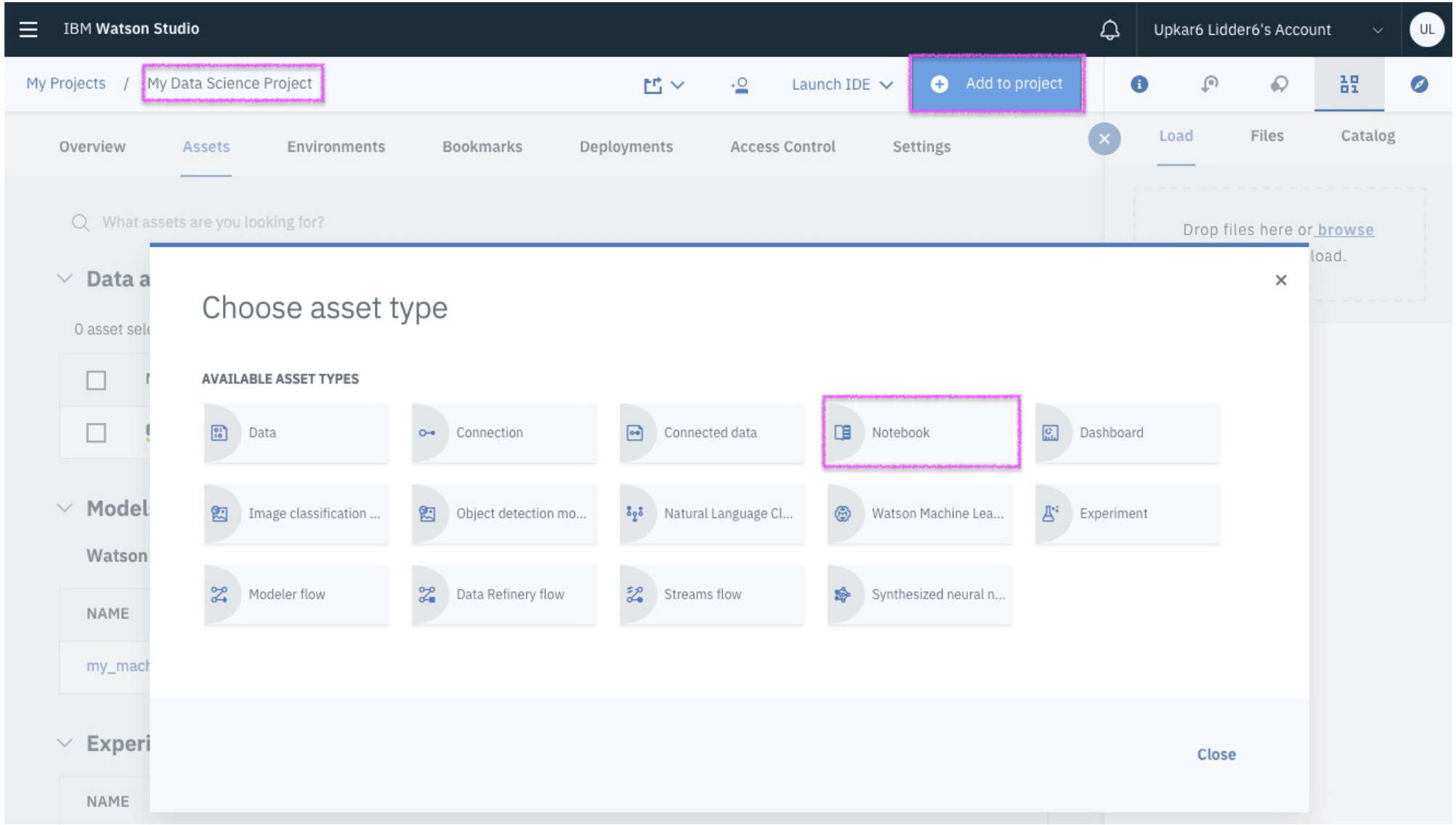
WML - deploy scikit-learn linear regression model

```
import json

# grab the model from IBM Cloud
published_model_uid = client.repository.get_model_uid(published_model)

# create a new deployment for the model
model_deployed = client.deployments.create(published_model_uid, "Deployment of scikit model")

#get the scoring endpoint
scoring_endpoint = client.deployments.get_scoring_url(model_deployed)
print(scoring_endpoint)

#use the scoring endpoint to predict house median price some test data
scoring_payload = {"values": [list(X_test[0]), list(X_test[1])]}
predictions = client.deployments.score(scoring_endpoint, scoring_payload)
print(json.dumps(predictions, indent=2))
```

WML - deploy scikit-learn linear regression model

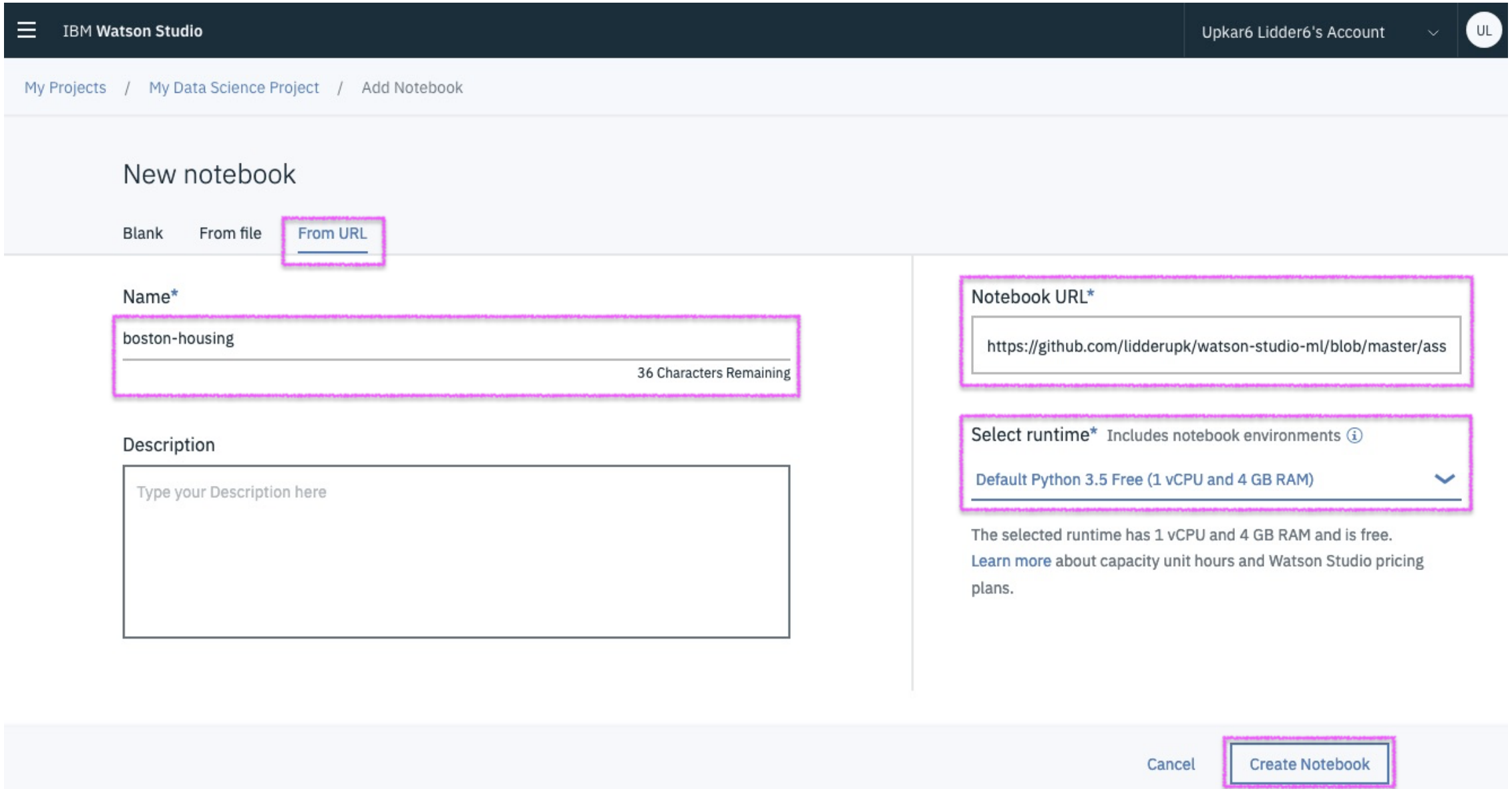
```
#########################################################################################

Synchronous deployment creation for uid: [redacted]' started

#########################################################################################


INITIALIZING
DEPLOY_SUCCESS


------------------------------------------------------------------------------------------------
Successfully finished deployment creation, deployment_uid='[redacted]'
------------------------------------------------------------------------------------------------


https://us-south.ml.cloud.ibm.com/v3/wml_instances/[redacted]/online
{
  "values": [
    [
      10.750284346772386
    ],
    [
      17.59893318342104
    ]
  ],
  "fields": [
    "prediction"
  ]
}
```

WML - try it out on your own !



http://bit.ly/waston-ml-sign

WML - create a new notebook from URL



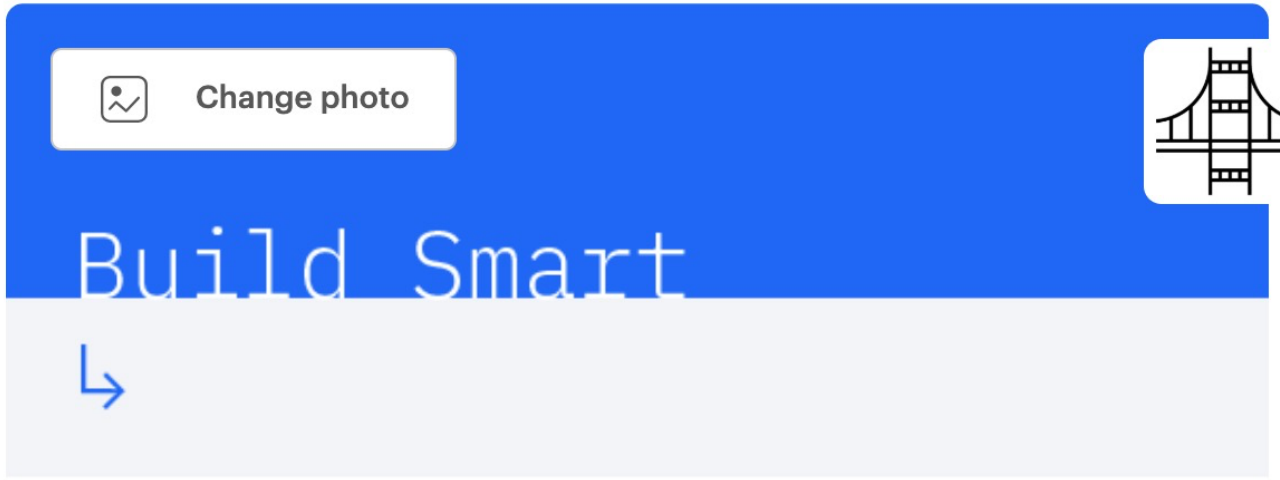
Grab the FULL URL from : http://bit.ly/boston-house-notebook

Thank you

Let's chat !

Upkar Lidder, IBM

@lidderupk
https://github.com/lidderupk/
ulidder@us.ibm.com



Part of **IBM Developer – 29 groups**

IBM Developer SF Bay Area

San Francisco, CA

7,164 members · Public group

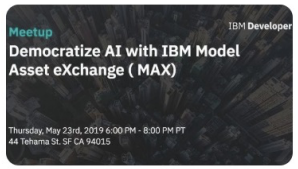
Organized by **Angie K** and **6 others**

IBM **Developer**

Share:

THU, MAY 23, 6:00 PM

Meetup: Democratize AI with IBM Model Asset eXchange (MAX)



44 Tehama St

Have you ever wanted to classify images, recognize faces or places in images, process natural language or text, or create recommendations based on time-series data in one of your applications? Join IBM Developer's meetup and learn how! With deep-learning (machine learnin...

61 attendees

Manage

Going

WED, MAY 29, 9:30 AM

Online Meetup: The what, the why, and the how of Knative on Kubernetes

Needs a location

RSVP on Crowdcast NOW! https://www.crowdcast.io/e/the-what-the-why-and-the ! Please do not forget to register on Crowdcast and join us using Chrome browser via Crowdcast on the event date! Wouldn't it be nice if as developers we could just focus on our code? That is the...

42 attendees

Manage

Attend

IBM Partners

Enabling Independent Software Vendors (ISVs) and tech companies for growth

Target audience

- ISVs and tech companies building and selling cloud solutions
- New to IBM Cloud
- Startups who aspire to build and sell their own solutions

Offers to help you get started

Build with up to $12,000 of free IBM Cloud™ credits ($1,000 per month for 12 months)

Integrate your solutions with leading-edge IBM Cloud technologies to deliver more innovation and value to your clients. Access more than **130 unparalleled services** including Watson™, Analytics and Security.

Build with 10TB of IBM Cloud Object Storage at no charge

Build data capability into your offering. IBM Cloud Object Storage is designed for high durability, resiliency and security.

Build with IBM Watson Assistant with a 1-year free trial

Receive access to 100K API calls per month plus 10 workspaces. Build and deploy chatbots quickly and efficiently with IBM Watson Assistant's advanced capabilities and seamless interface.

Get started

Experience IBM's countless partner benefits. Start building and selling with IBM today.

Learn more and access offers at ibm.com/partners/start

Build with IBM Cloud Kubernetes Service with a 1-year free trial

Containerize your solution with 1TB of block storage. Ship all your applications in one agile, well-defined structure with IBM Cloud Kubernetes Service.

Build with IBM Blockchain with a 6-month free trial

Build a network with up to 3 organizations to prototype. Build a secure business transaction network for your clients using blockchain and smart contracts.

Finished building and testing? Go-to-market with IBM

Access Provider Workbench, attend an orientation session and join the premier network of over 400 partners who are already listing their solutions on the IBM Marketplace.

Is your business a Startup? Build with up to $120,000 in IBM Cloud credits

If your business revenue in the last 12 months is less than $1M and you've been in business for fewer than five years, then you may qualify for Startup with IBM.