# Introduction to Streamlit

Estimated time needed: **45** minutes

## Objectives

After completing this lab, you will be able to:

* Describe the core concepts of streamlit framework
* Build your first Streamlit app

# Lab environment setup and preparation

If you plan to develop the app in your local development environments, you need to make sure you have the following:

* Your favorite IDE or text editor. Note we are developing a Python app with script

files so notebooks like Jupyter Notebook is not very suitable for developing app.

* Python 3.7 - Python 3.9
* PIP
* We recommend you create a virtual environment for building the Streamlit apps.
* Click [here](https://docs.python.org/3/library/venv.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMML321ENSkillsNetwork32585014-2022-01-01#module-venv) to learn more about Python virtual environments.
* Download the sample app from here:

<https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML321EN-SkillsNetwork/labs/module_5/first_app.zip>

## Install required Python libraries and test the app

* In your terminal, make sure you have Python installed or activated any Python virtual environments.

First cd to the unzipped first\_app directory:

cd first\_app

and install required Python libraries specified in requirements.txt file for this app:

pip3 install -r requirements.txt

First, let's run the first app by the following command:

streamlit run app.py

If you are on your local environment,

* Open your browser and enter URL <http://localhost:8501>

If you are using Skills Network Labs environment (TBD):

# Introduction to Streamlit

Streamlit is an open-source app framework which allows data scientists and machine learning engineers to quickly build interactive machine learning and data analytics web apps. It is so easy to use that you could turn your Python data and modeling scripts into shareable web apps in minutes, and no web front‑end experience required at all.

## Core concepts

Streamlit allows you to write apps the same way as you write Python scripts.

* Streamlit reruns your entire Python script from top to bottom if you updated your source code or there is

a user interaction (like clicking a button)

* Streamlit can easily display data, charts, maps, which is similar to Jupyter notebook.
* Streamlit provides a set of UI Widgets like text label, button, slider, checkbox, selectbox, which you can use

to interact with your data and models

* Streamlit uses cache to avoid recomputing heavy functions such as loading large csv files.
* Streamlit supports multiple cloud deployment methods. Once you have built and tested a Streamlit app, you can easily deploy it on many cloud environments to showcase your

work

## A simple example app

Now let's build a very simple Streamlit app together to better understand how Streamlit works.

* Open the app.py file and you can see an empty script file with only two imports streamlit and pandas
* We can now run the empty app to see. In the terminal, run the following command:

streamlit run app.py

and you should see an empty page.

* Then under comment #1, copy the following code snippet to add a Streamlit title widget:

st.title("This is a sample app")

* Refresh the page or click the rerun button on the page, and you should see a newly added title
* Under comment #2, copy the following code snippet to add a button title widget:

button1 = st.button("Click to show a dataframe")  
print(button1)

As you can see in the console, the value of button1 is False which means it is not clicked. and if we click the button, The Streamlit app will re-run and the button1 becomes True

button1 = st.button("Click to show a dataframe")  
print(button1)

Then, we can add the following code snippet to show a simple Pandas dataframe if button1 is clicked:

if button1:  
 df = pd.DataFrame({  
 'column1': [1, 2, 3, 4],  
 'column2': [10, 20, 30, 40]  
 })  
 # Show the Pandas dataframe using st.dataframe() method  
 st.dataframe(df)  
 # Visualize the column1 series using st.line\_chart() method  
 st.line\_chart(df['column1'])

Refresh the page and click the button, and you should see a dataframe is displayed as a table using st.dataframe() method and visualized as a line chart using st.line\_chart()

* Next, under comment #3, let's add two sliders to receive some numerical values from users:

slider1 = st.slider("Slider1", min\_value=1, max\_value=10, value=1)  
print(slider1)  
slider2 = st.slider("Slider2", min\_value=1, max\_value=10, value=2)  
print(slider2)

* Refresh the page and we can see two sliders are added. From the console, we can see the value of slider1

is 1 and value of slider2 is 2. If we scroll the sliders, their values will be updated accordingly (shown in the console).

* Lastly, under comment #4, we can add a text label to show the sum of two sliders:

# Create a streamlit subheader widget  
st.subheader("The sum of slider1 and slider2 is: ")  
st.text(slider1 + slider2)

# Reference

You can check more details in here:

* [Streamlit documentation](https://docs.streamlit.io/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMML321ENSkillsNetwork32585014-2022-01-01)

# Summary

By now, you should have some basic understanding about how Streamlit works. Next, we can take a look at our course recommender app.

# Build a course recommender app with streamlit

Estimated time needed: **60** minutes

## Objectives

After completing this lab, you will be able to:

* Understand and run the provided skeleton recommender app
* Extend the recommender app by completing model training and prediction methods

# Lab environment setup and preparation

If you plan to develop the app in your local development environments, you need to make sure you have the following:

* Your favorite IDE or text editor
* Python 3.7 - Python 3.9
* PIP
* We recommend you create a virtual environment for building the Streamlit apps.
* Click [here](https://docs.python.org/3/library/venv.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMML321ENSkillsNetwork32585014-2022-01-01#module-venv) to learn more about Python virtual environments.

# Personalized Course Recommender app

Download and unzip the provided recommender app template:

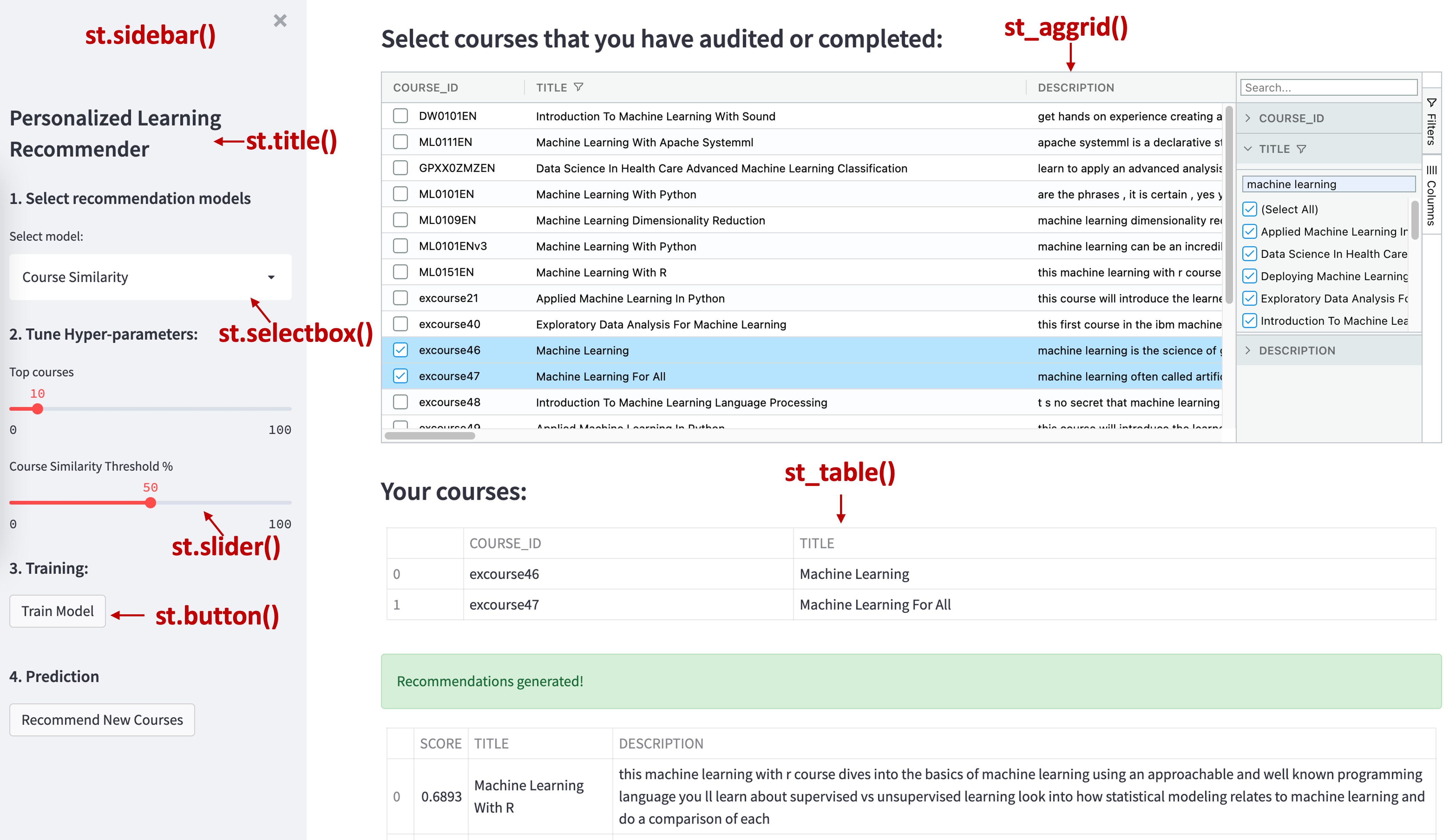
wget <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML321EN-SkillsNetwork/labs/module_5/app.zip>

unzip app.zip  
rm app.zip

In the unzipped app folder, we can see the following files and directories:

* data folder contains related datasets such as ratings, course content, similarity matrix, etc.
* backend.py contains backend machine learning functions such as loading datasets, model training, predictions, etc.
* recommender\_app.py is the main Streamlit Python script mainly implements the user interactions with the backend machine learning code
* requirements.txt specifying the required Python libraries for the app. You can add more libraries whenever needed

If we run the sample app, you should see the following page pops-up:



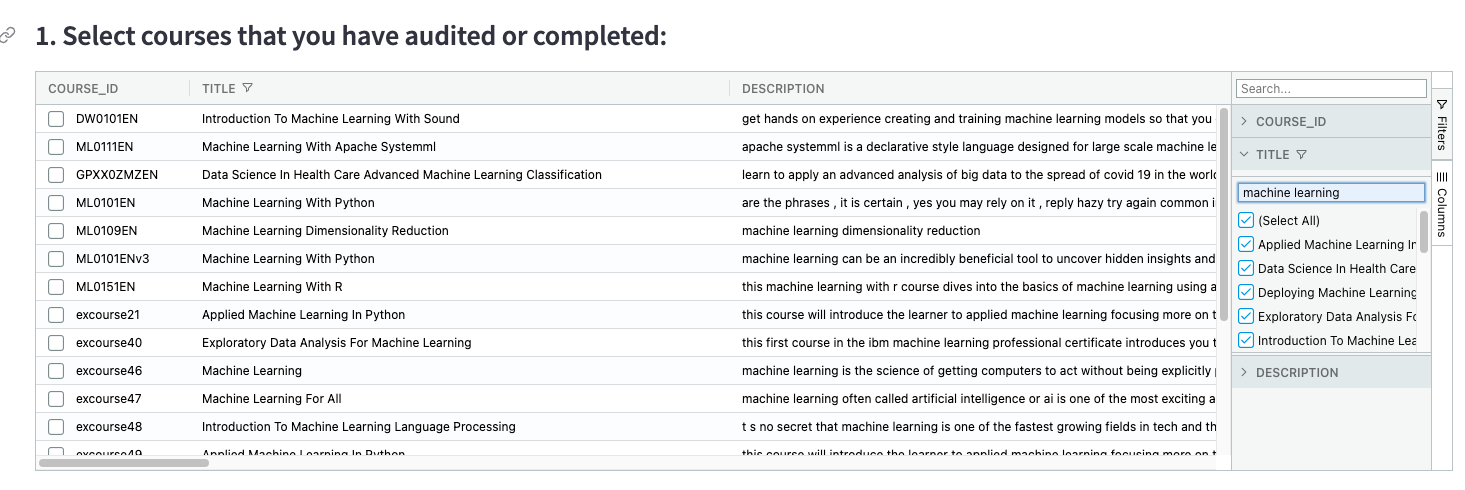
We also labelled the Streamlit widgets on the above screenshot, and you can check their details in the source code.

If take a look at the course selection table, here we used a streamlit-aggrid plugin to create a st\_aggrid() widget to better interact with the dataframes. For example, you can easily select, filter, and search the courses you want.

The user interactions in this app are pretty straightforward:

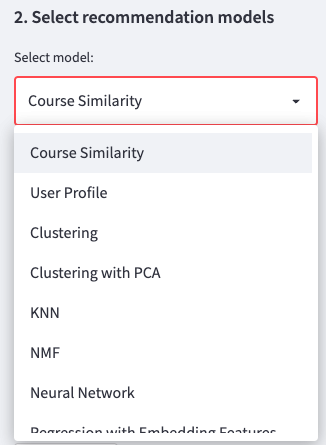
1. As a test user, you first need to search and select the courses you have audited or completed. You can filter courses

via the Filters and Columns menu items on the right side of the aggrid.



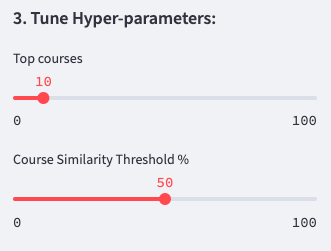
1. Then, once you have created a course enrollment list, you can choose which recommender model you want to use to

generate course recommendations. Here in this sample app, we only provided the course similarity based model for demo purpose. You will need to port the model implementation from the notebooks you have completed in the previous labs



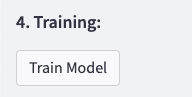
1. Depends on your model, you may add different hyper-parameters UI widgets and determine the hyper-parmeters

for training and prediction



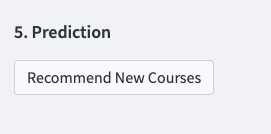
1. Depends on your model, you may need to re-train it with new data been added in Step 1.

You will need to port the model training code from the notebooks you have completed earlier.



1. Once you have your model trained, you need to determine the test data for the model to make predictions.

You need to figure out all the unseen/unselected for the test user and estimate the rating using the model.



# Extend the recommender app by completing model training and prediction methods

Now you can extend this app by porting the model training and prediction code you have done in the previous notebooks. More specifically, you need to modify three places in the app source code:

1. Add more hyper-parameters widgets.

In the recommender\_app.py, find the code area below comment # Hyper-parameters for each model. This is the area where we render different UI widgets for receiving hyper-parameter values.

For example, we added two sliders bar for only showing the top courses and the threshold to determine if two courses are similar or not.

params = {}  
# Course similarity model  
if model\_selection == backend.models[0]:  
 # Add a slide bar for selecting top courses  
 top\_courses = st.sidebar.slider('Top courses',  
 min\_value=0, max\_value=100,  
 value=10, step=1)  
 # Add a slide bar for choosing similarity threshold  
 course\_sim\_threshold = st.sidebar.slider('Course Similarity Threshold %',  
 min\_value=0, max\_value=100,  
 value=50, step=10)  
 params['top\_courses'] = top\_courses  
 params['sim\_threshold'] = course\_sim\_threshold

All selected parameters will be stored in a Python dictionary called params and be passed into training and prediction functions.

params = {}

1. Add model-specific training code. If you find the code area under # Training UI, you should see the

following code snippet:

st.sidebar.subheader('4. Training: ')  
training\_button = st.sidebar.button("Train Model")  
training\_text = st.sidebar.text('')  
# Start training process  
if training\_button:  
 train(model\_selection, params)

Basically, what it does is when you click the training\_button button, it calls the backend.train() method via the train() method with a model name and hyper-parameters.

Depends on which model is selected, the actual training code can be different.

1. Add model-specific prediction code. If you find the code area under # Prediction UI, you should see the

following code snippet:

# Prediction UI  
st.sidebar.subheader('5. Prediction')  
# Start prediction process  
pred\_button = st.sidebar.button("Recommend New Courses")  
if pred\_button and selected\_courses\_df.shape[0] > 0:  
 res\_df = predict(model\_selection, params)  
 st.table(res\_df)

Basically, what it does is when you click the pred\_button button, it calls the backend.predict() method via the predict() method with a model name and hyper-parameters.

Depends on which model is selected, the actual prediction code can be different.

# Reference

You can check the following links for more details about Streamlit:

* [Streamlit documentation](https://docs.streamlit.io/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMML321ENSkillsNetwork32585014-2022-01-01)
* [streamlit-aggrid](https://github.com/PablocFonseca/streamlit-aggrid)

# Summary

In this lab, you have learned how to extend the recommender template app to include more recommendation models you have built in the previous notebooks.

In addition, you are encouraged to modify and improve the template app to better serving and demonstrating your recommendation models.