Project Scenario

Sound Realty helps people sell homes in the Seattle area. They currently spend too much time and effort on estimating the value of properties. One of their staff has heard a lot about machine learning (ML) and has created a basic model to estimate the value of propeties. The basic model uses only numeric variables and ignores some other attributes. Despite the simplicity of this model, the folks at Sound are impressed with the proof of concept and would now like to use this model to streamline their business. They have contracted us to help deploy that model for broader use. Our job is to create a REST endpoint that serves up model predictions for new data, and to provide guidance on how they could improve the model.

Sound Realty has provided a Python script (`create_model.py`) which trains the basic model. The data used to train the model is in the `data` directory, which includes:

- `data/kc_house_data.csv` – Data for training the model. Each row corresponds to a sold home. `price` is the target column, the rest can be used as features if appropriate.
- `data/zipcode_demographics.csv` – Additional demographic data from the U.S. Census which are used as features. This data should be joined to the primary home sales using the `zipcode` column.
- `data/future_unseen_examples.csv` – This file contains examples of homes to be sold in the future. It includes all attributes from the original home sales file, but not the `price` , `date` , or `id` . It also does not include the demographic data.

The model developer at Sound Realty used a Conda environment to create the model, which has been captured in Conda's YAML format. Assuming Conda has been installed in your environment, you can recreate that environment with the following:

```
conda env create -f conda_environment.yml
# Activate the environment.  Repeat for newly spawned shells/terminals
conda activate housing
```

Once you've created and activated the environment, you can run the script which creates the model:

```
python create_model.py
```

The model artifacts will be saved in a directory called `model/` with the following contents:

- `model/model.pkl` – The model serialized in Python Pickle format.

- `model/model_features.json` – The features required for the model to make a prediction, in the order they were passed during training.

Deliverables/Requirements

1. Deploy the model as an endpoint on a RESTful service which receives JSON POST data.
   - The inputs to this endpoint should be the columns in `data/future_unseen_examples.csv` .
   - The endpoint should return a JSON object with a prediction from the model, as well as any metadata you see as necessary.
   - The inputs to the endpoint **should not** include any of the demographic data from the `data/zipcode_demographics.csv` table. Your service should add this data on the backend.
   - Consider how your solution would scale as more users call the API. If possible, design a solution that allows scaling up or scaling down of API resources without stopping the service. You don't have to actually implement autoscaling, but be prepared to talk about how you would.
   - Consider how updated versions of the model will be deployed. If possible, develop a solution that allows new versions of the model to be deployed without stopping the service.
   - Bonus: the basic model only uses a subset of the columns provided in the house sales data. Create an additional API endpoint where only the required features have to be provided in order to get a prediction.
2. Create a test script which submits examples to the endpoint to demonstrate its behavior. The examples should be taken from `data/future_unseen_examples.csv` . This script does not have to be complicated, only needs to demonstrate that the service works.
3. Evaluate the performance of the model. You should start with the code in `create_model.py` and try to figure out how well the model will generalize to new data. Has the model appropriately fit the dataset?
4. Improve the model by applying some basic machine-learning principles. We're not interested in in getting the absolute best predictive performance for the model, don't devote too much time to this step. This is not a Kaggle competition. Rather we're interested in your understanding of data science concepts and your ability explain the decisions you made.

Recommendations

- We recommend using Docker to containerize and deploy the model. However, feel free to use a different technology if there is one you are more familiar with.
- In addition to Docker, you'll need to use several other components to create a scalable REST API. Google is your friend here, do some web searching to figure out what other components are needed to deploy a scaleable REST API for a Python application.
- For the updated model use a traditional machine learning algorithm (no need for deep learning). Build out an 80% solution.

Non-Requirements

- **Completing in a specific amount of time.** Life is busy and chaotic. We understand you will not be able to work full time on this project.
- **Running at scale.** Everything can be done on a laptop running Docker Desktop, there is no need to deploy your code to a cloud service or cluster.
- **An exact end result.** Two candidates given this assignment will find different solutions. Feel free to choose your own adventure as long as the base requirements are met.

Time management

We cannot stress these two enough:

1. Build the simplest possible solution first, utilizing tools you are familiar with when possible.
2. Don’t get stuck on one aspect of the project. Ask questions and use the internet for research. Focus on your core strengths.