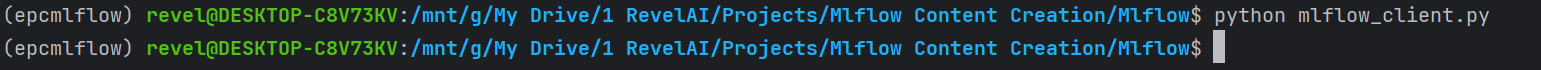
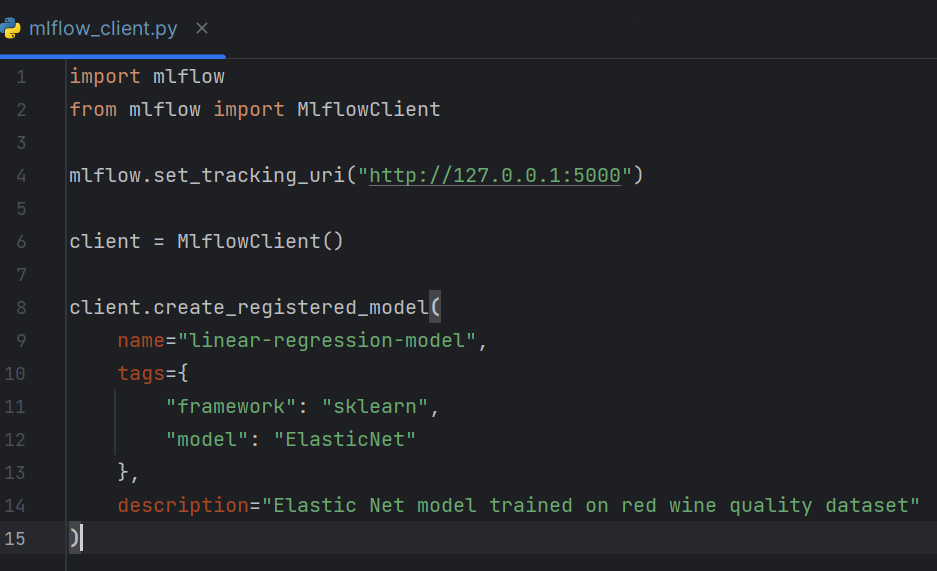
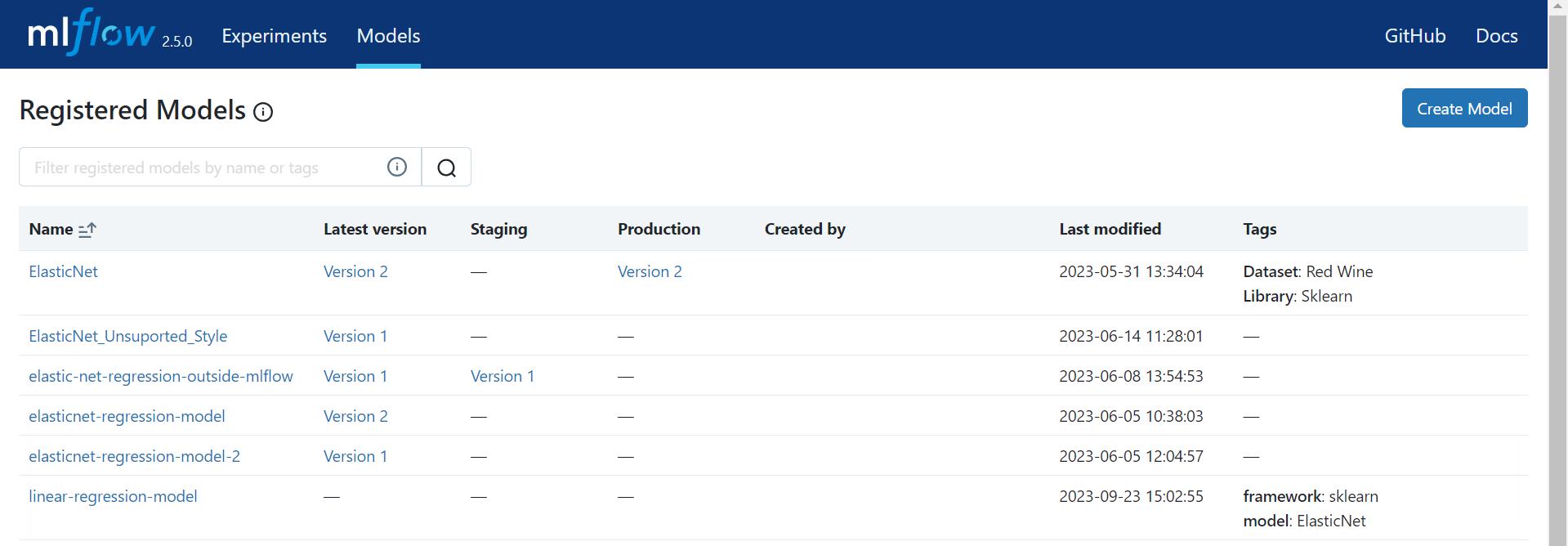
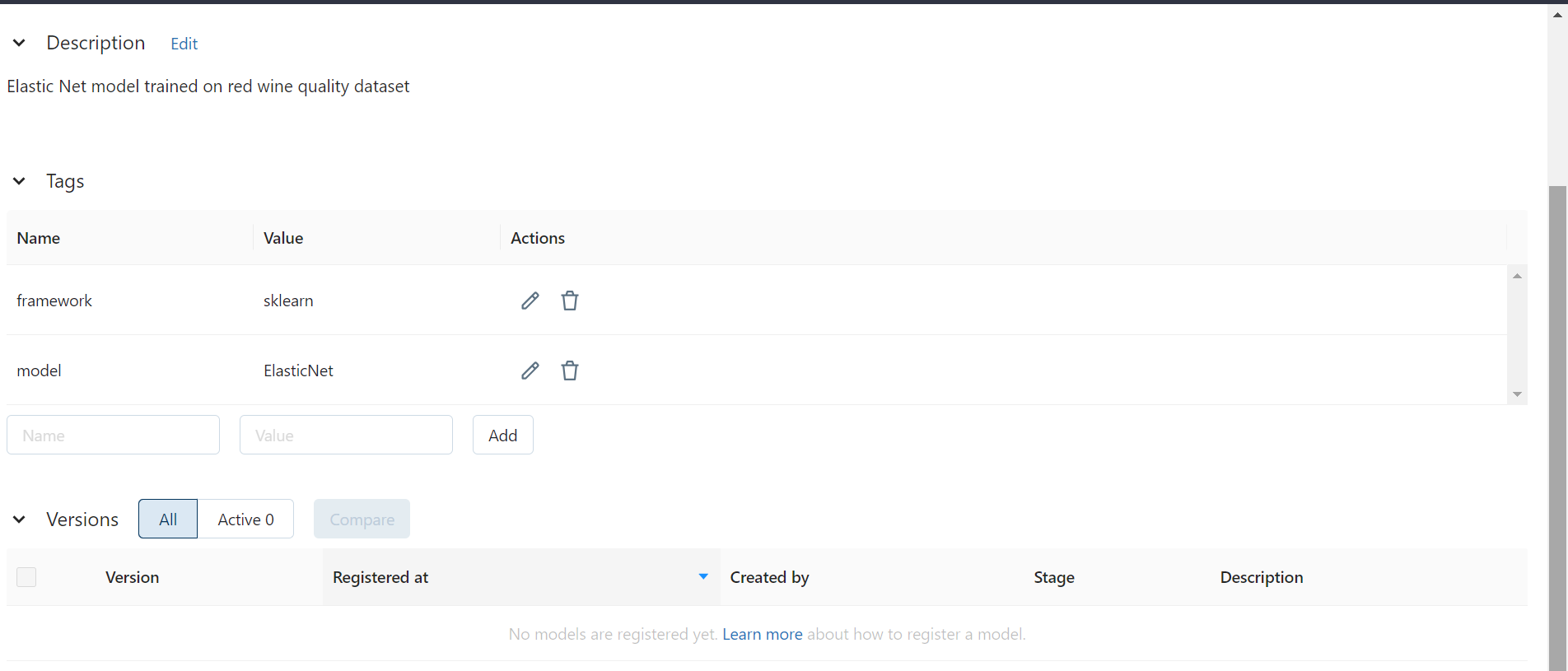
Hey, Welcome to this lecture. In this lecture we’re going to be looking at Model Versioning and management. Basically we would be playing around with the function of mlflow client for model registry and versioning. You remember when we’re in Mlflow Model Registry sections we skipped a lot of function, the api interface function, that we’ll cover them in mlflow section. Well now’s the time. A little recap of Model Registry and versioning. Model Registry is an mlflow component which is a centralized model store, containing set of APIs such as this and UIs which we have already talked, to collaborate manage the full lifecycle of an Mlflow Model. It provides model lineage, model versioning, stage transitions, and annotations. What happens is that during experimentation you log a model. And you compare all your experiments and runs, and find few of the best models. You then register those model unique name, containing versions, associated transitional stages, model lineage, and other metadata, more like putting them aside with labels on them. So you deployment team could look into those models and their tags and deploy the model with the suited tag. In this whole lecture, we would registering the models, versioning them, retrieving, deleting, updating and searching your suitable models. Without wasting anytime let’s get our hands dirty in it.

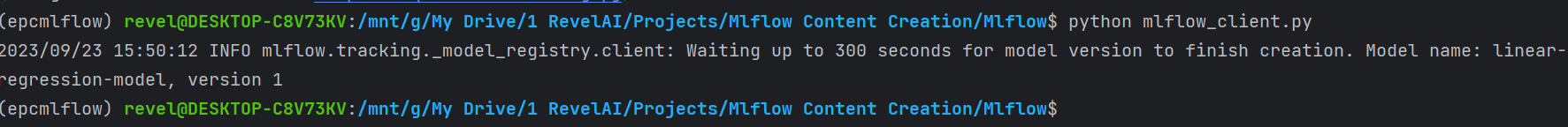
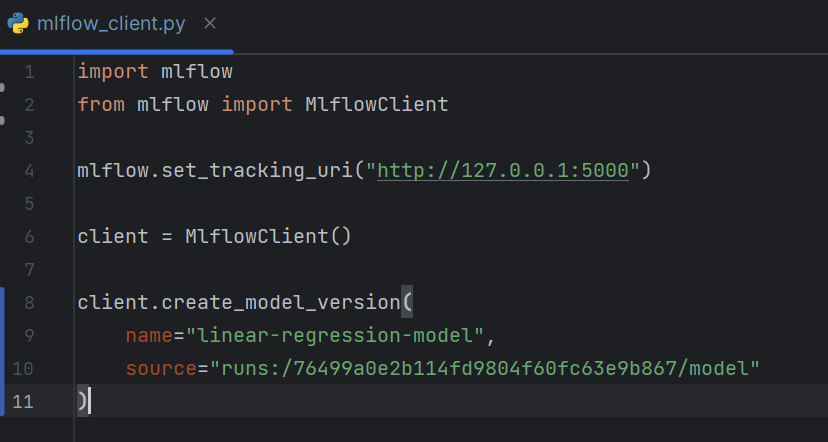
To kick start I’m going to first register the model. Removing all the previous code till client. And also the some of the imports that we don’t need. Ok so we need a model to register but let me first introduce you with the create\_registerd\_model function that’s used to register the model. This function takes in three parameters, name you wanna give to the registered model, the tags you wanna include, which are in the form of dictionary key value pairs and the description of the model. The function returns a single object mlflow.entities.model\_registry.RegisteredModel which is created by backend. Remember this function is gonna create an empty registry with no model in it. Let’s use this function and create an empty registry. Client.create\_registered\_model, name = “linear-regression-model”, tags to framework sklearn and model ElasticNet that’s just me tagging, you can pick up your own specific tags depending upon your needs. Then the description, I’m adding "Elastic Net model trained on red wine quality dataset". Again chose the best description that suits your needs. Let’s run it.



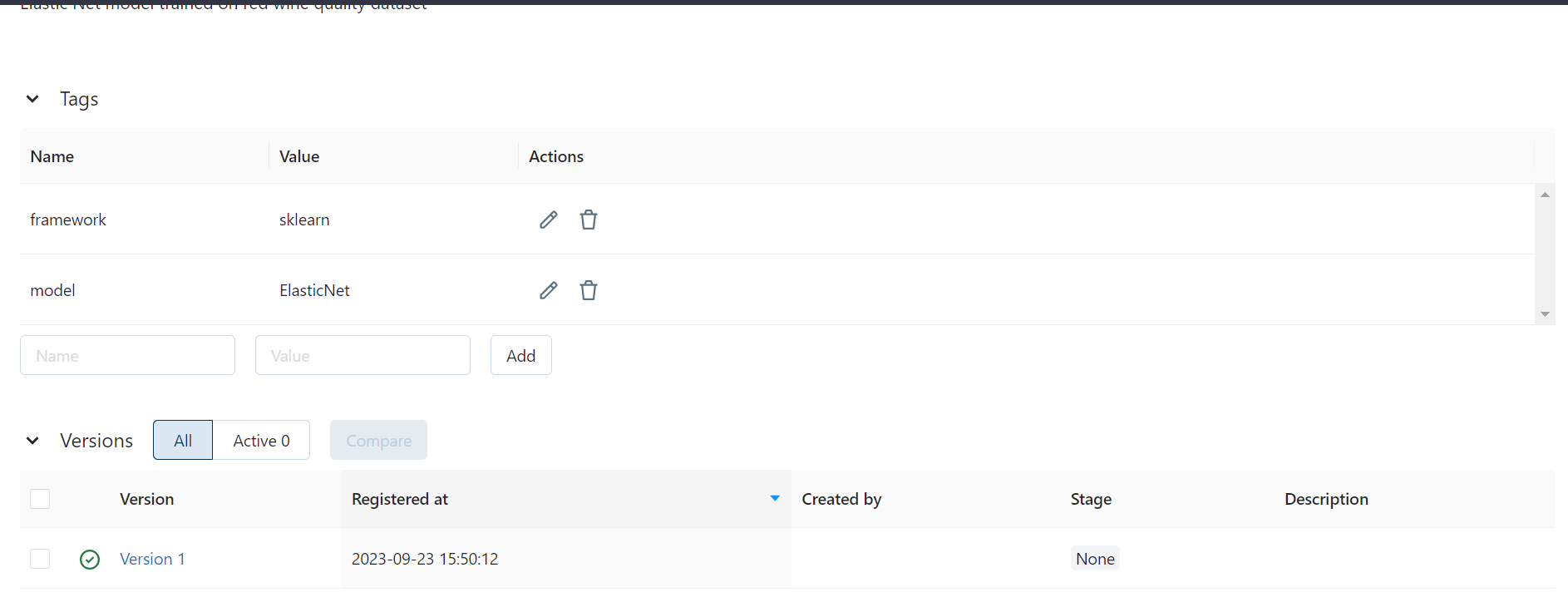
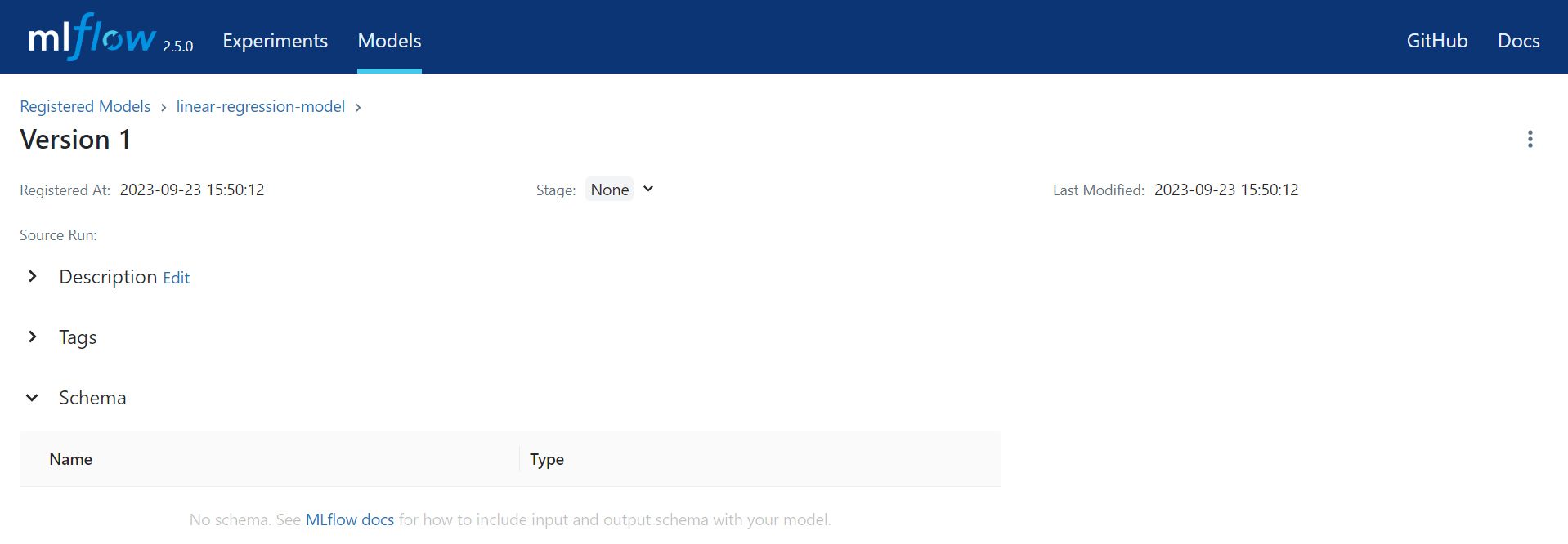
Let’s go to UI and see it there. Moving on models section. We can see our model registery there. Opening linear-regression-model. You see its empty. Let’s add a model to it.

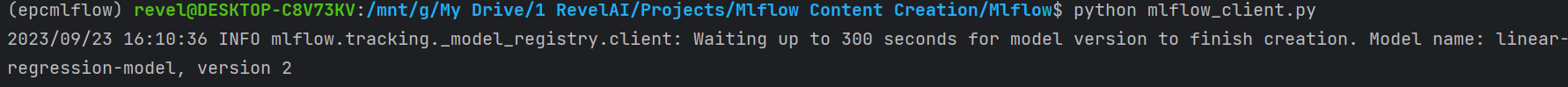
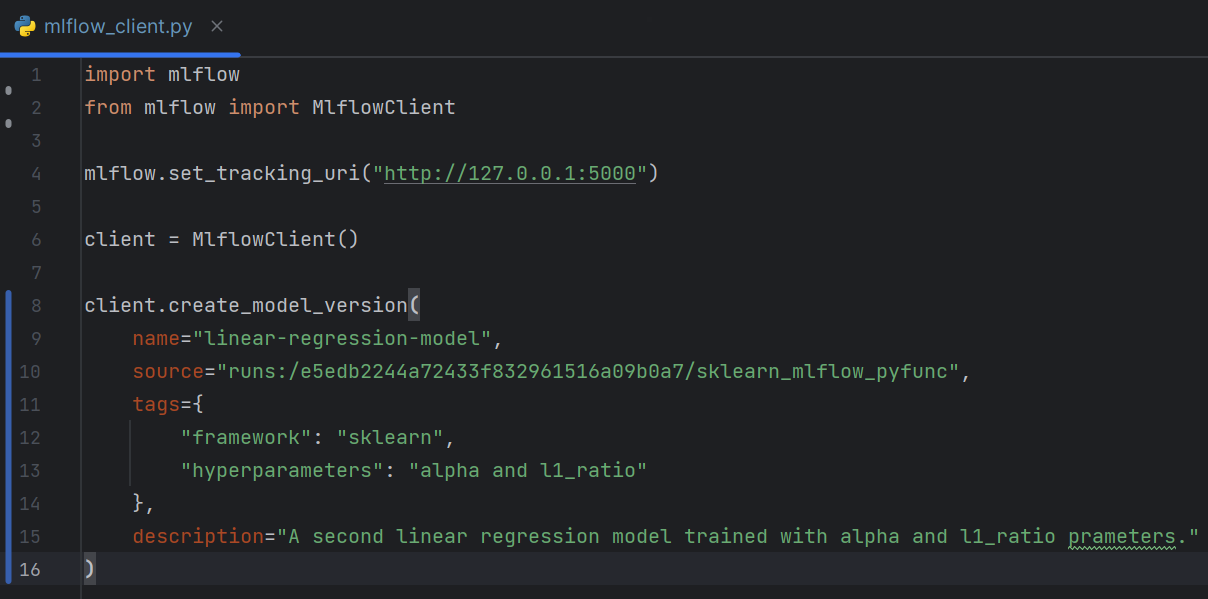
In order to add a model to it or create a model version in mlflow language. We need to use create\_model\_version function. This function takes in seven parameters, name of the model, source where the model is, the run\_id the ID of run which logged that model, the tags, run\_link the link to the run from an MLflow tracking server that generated this model, description of the version that you wanna give, await\_creation\_for which are Number of seconds to wait for the model version to finish being created and is in **READY** status. By default, the function waits for five minutes. Specify 0 or None to skip waiting. This function also returns a single object of mlflow.entities.model\_registry.ModelVersion which was created by backend. Let’s use this function now. Client.create\_model\_version name which is same as the name of model registry and source equals runs then the ID of the run where the logged model is. I want you to just go to any previous runs and grab its and name of the model. Its because we don’t have to create an example and log the model and then register it. We logged several model behind. Let’s just pick up any. I’m pasting the ID I grabed of one of the runs of my 17th experiment. The model name was model itself. Remember you need to grab the exact name itself. A different name will cause error. Next comes run\_id and few other parameters but we don’t need to specify any of these. We have already specified the ID so we don’t need to provide again. Fun fact you can provide the complete uri to the model or just the runs:/run-id/model-name both works well. You can’t only specify model-name and run-id otherwise it would give errors. Remember the above expression creates an empty register and this expression creates a model version inside of it. Running code as it will give you weird error. So you’ve to remove above expression because its try to create an empty which is already there and that is something that you can’t do. So let’s just remove it. Lastly if you think create\_model\_version can work without a registry that it would automatically create the registry and add a model version to it. Than you might be wrong. Its not gonna work like that. So with that being said let’s just run the code and add the model version to our empty registry.



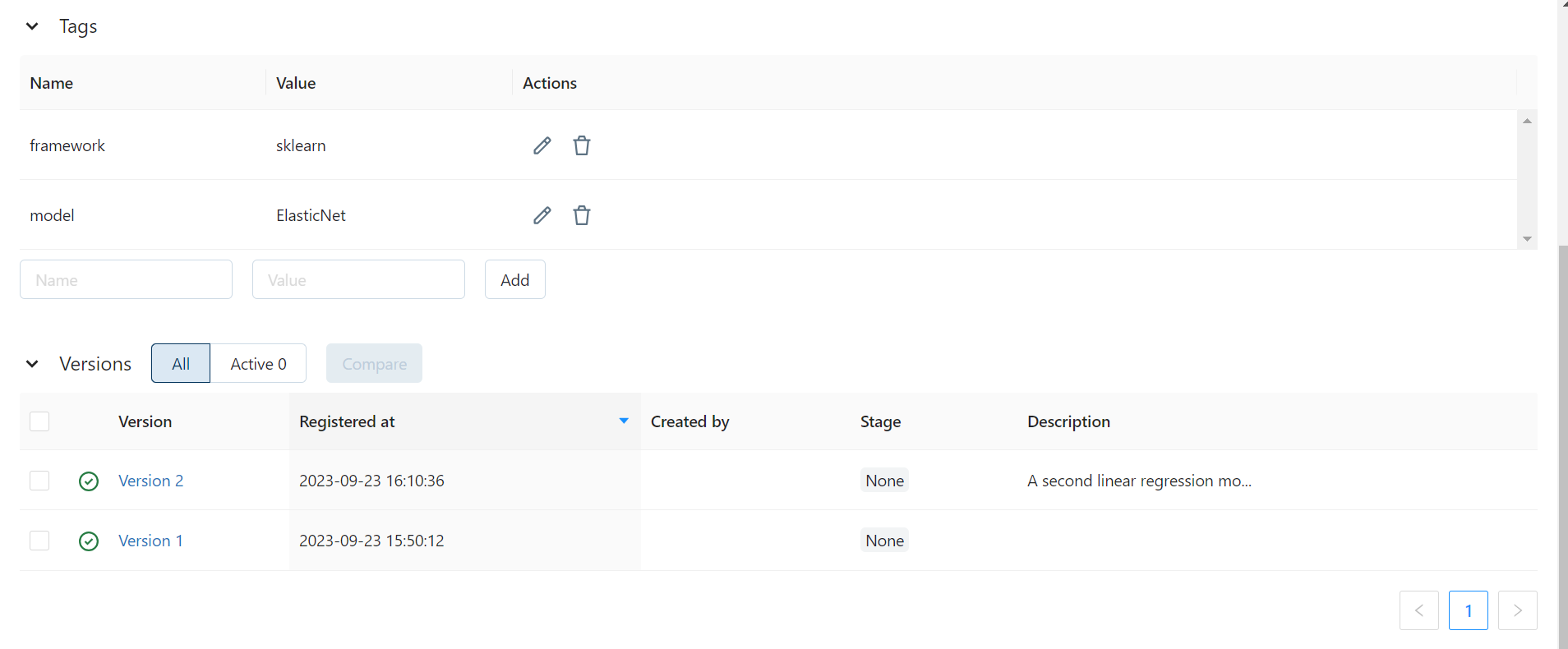
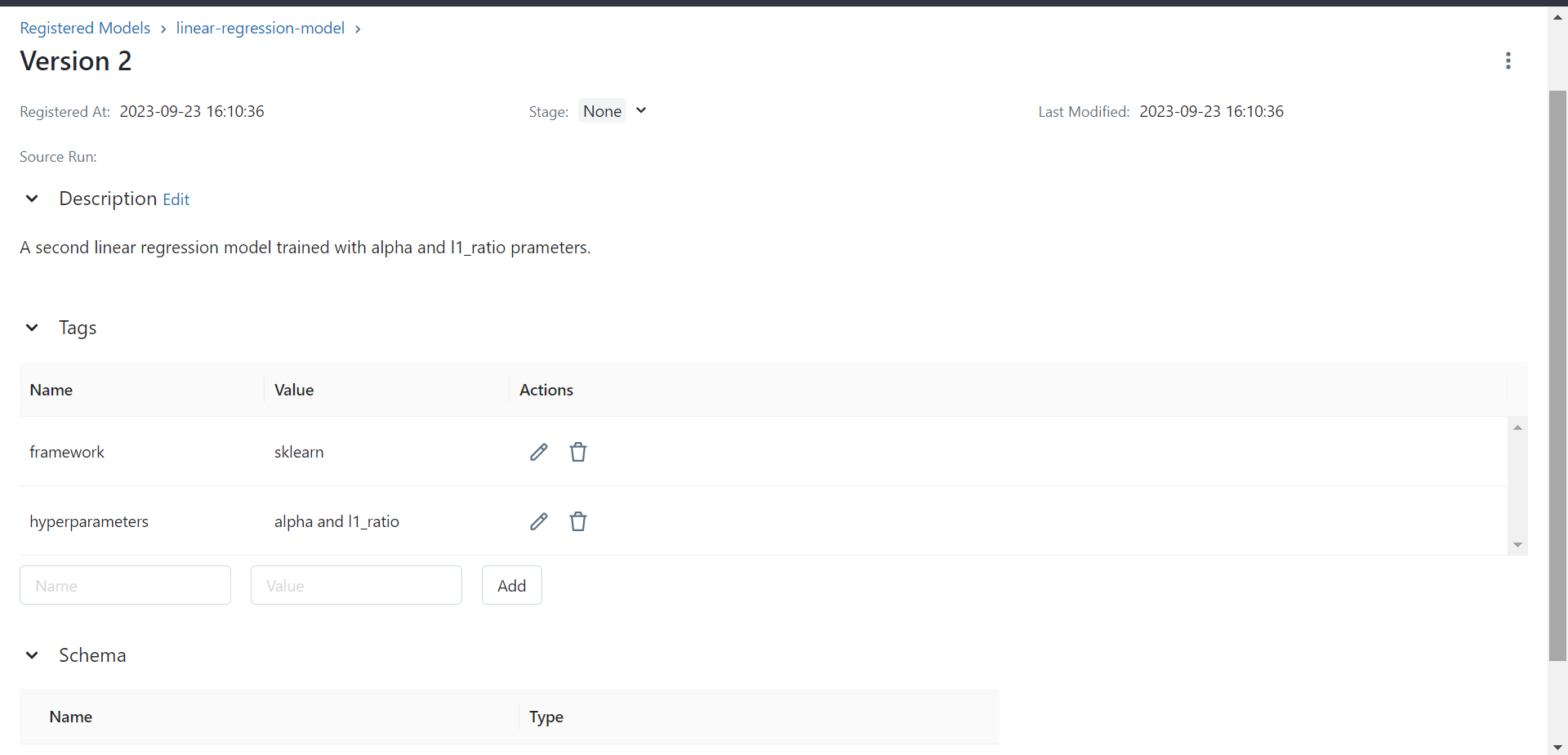
Ok! it confirms that the model version has been added, let’s also check this on UI. Moving to UI. Refreshing it. You can see its there. Its no longer empty its got a model version now. We can look into it, which doesn’t contain much of the data because we didn’t added any.

Let’s add second version to it. A different model. Let’s add a model from 15th experiment of mine. Grabing its ID, replacing the ID with new ID, and replacing the model name with name present in the run sklearn\_mlflow\_pyfunc. And tags, framework be sklearn and hyperparameter be “alpha and l1\_ratio”. And we can add description of like “A second linear regression model trained with alpha and l1\_ratio prameters.". Let’s run it.



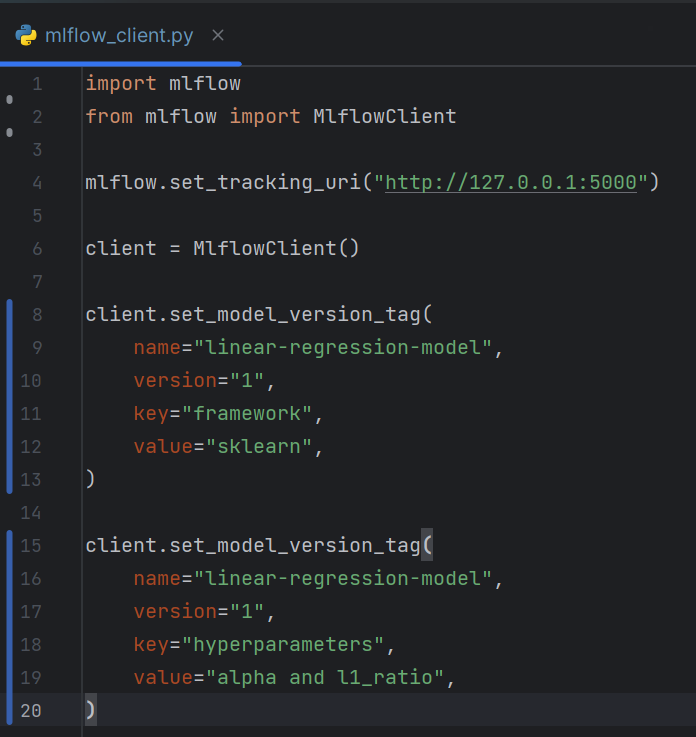
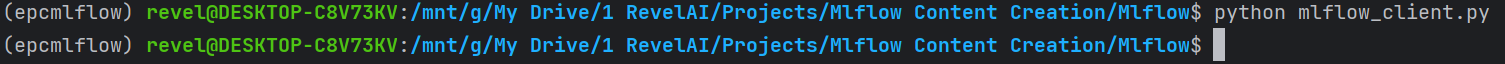
Let’s check this in UI. Models, registry and then there’s our version 2. Opening it up. And its got all the info we added.

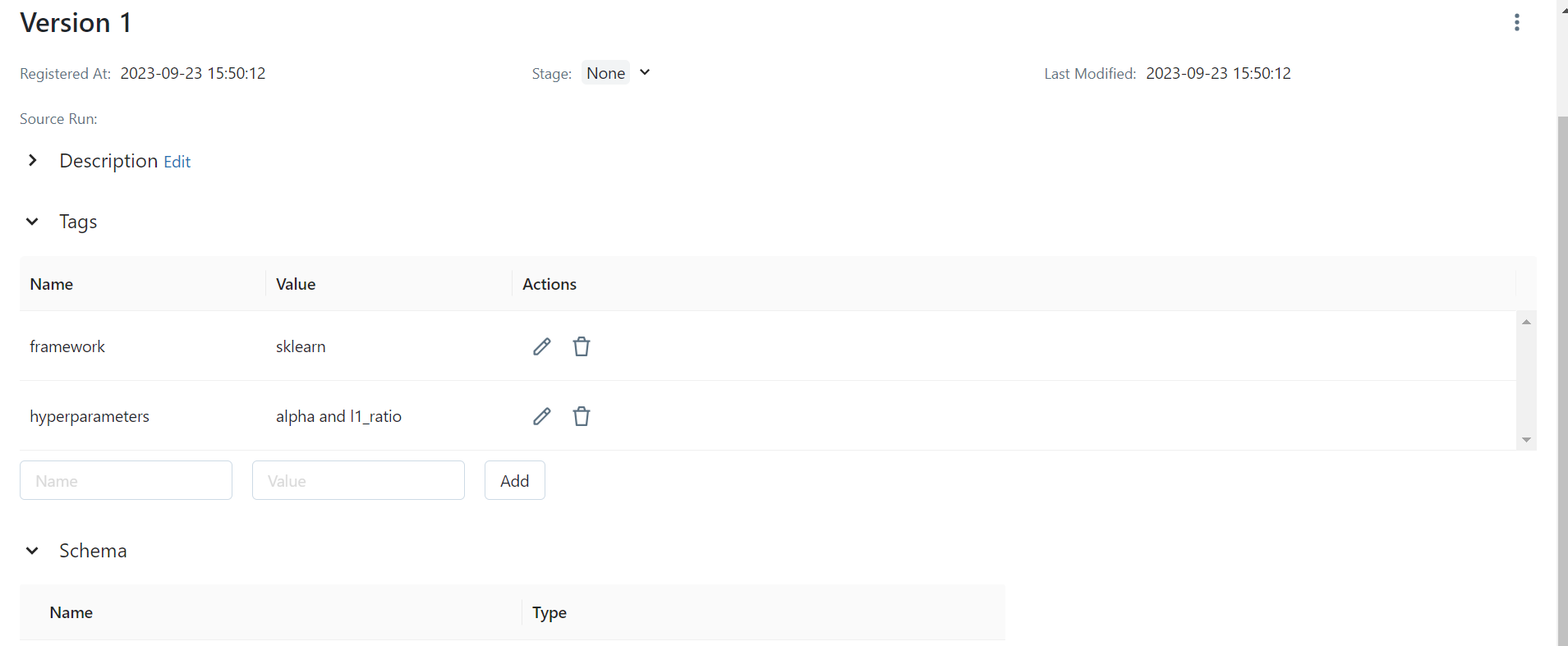
 

Guys, you probably noticed by now. All I’m doing is something we’ve done before, the only difference is now I’m doing mlflow client. So it should be pretty obvious to you by now.

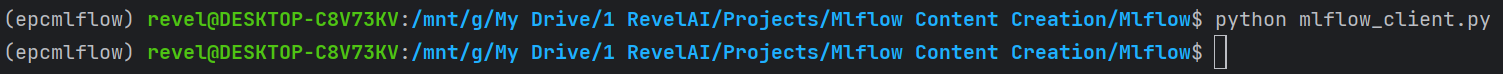
Now as you know the model version 1 didn’t have any tag associated to it. Let’s just set tags to it. The function for that is set\_model\_version\_tag. This function sets tag to the model version. It takes in five parameters, name of the registered, version of the model, key and then value, lastly the registered model stage. Stage is something we’ve not worked yet. A little distinction is that you either set tag to specific version of the model or all the models with the specific. You can’t specify both these parameters together. I’m only going to work version for now. We will work stage later on. This function returns nothing. It simply sets the tag to the model with specific version or stage. Let’s use it. Client.set\_model\_version\_tag, name equals linear-regression-model, then the version. I’m specifying tag to version 1 so “1” in quotation. Then key be “framework” and value be “sklearn”. I wanna add two tags. But this function only allows for one tag at a time. Let’s repeat this function and set tag “hyperparameters” to be “alpha and l1\_ratio”. Let’s run the code and add tags to version 1.

It worked well. Let’s also check this on UI. Moving UI, models, entering into registry, opening version 1, its got our two tags there.

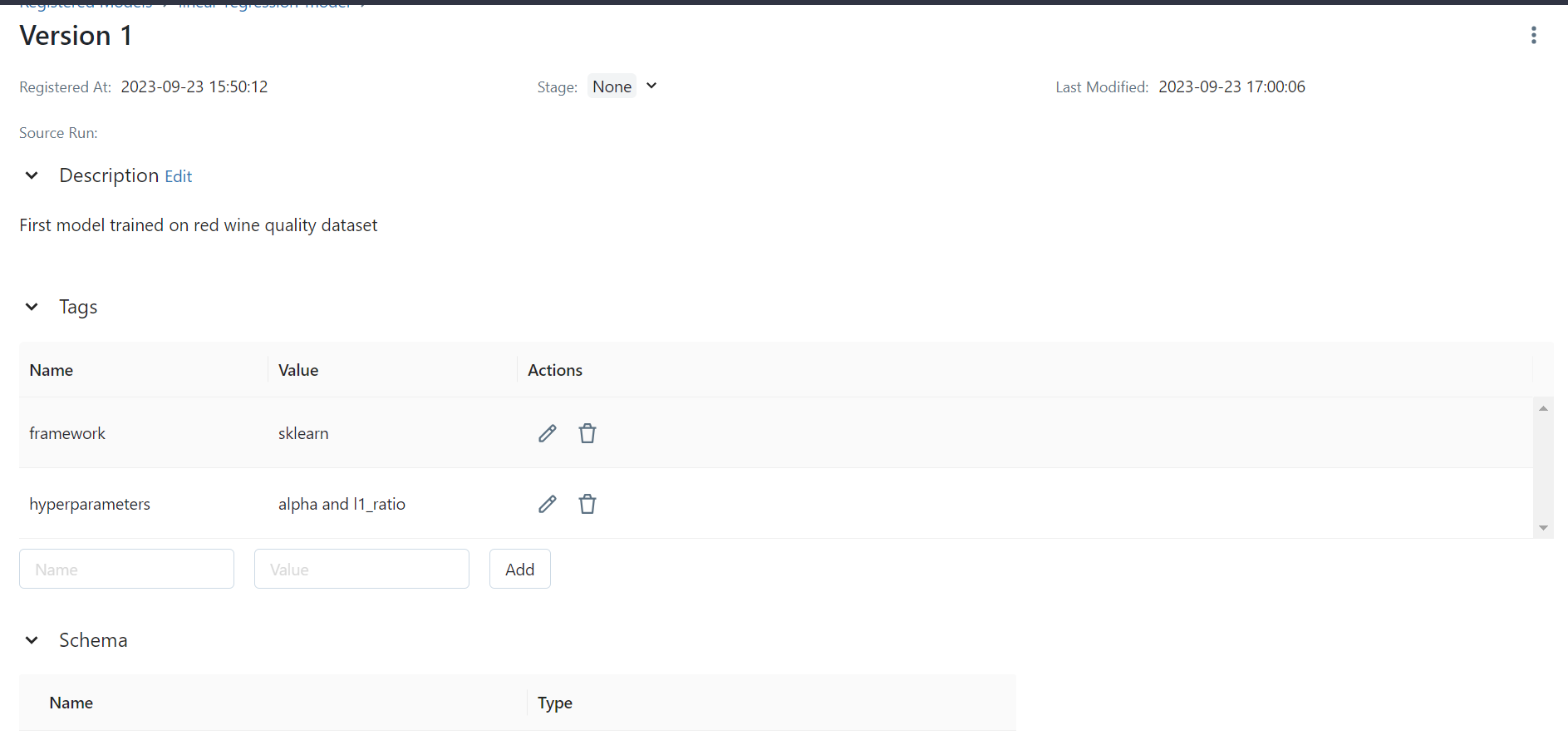
 



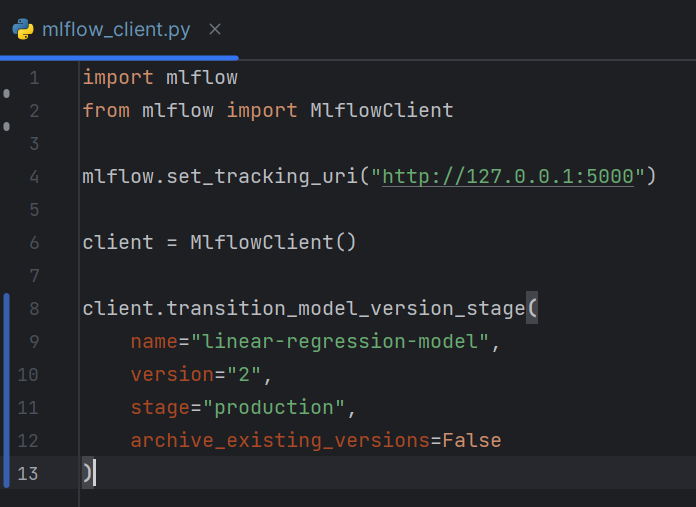
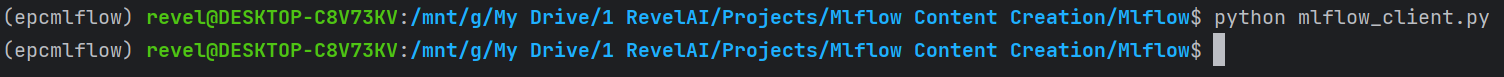
Now model version 1 also don’t have any description we can add some description to it as well. The update\_model\_version function is used to update metadata of the model version. Currently it only update description. We didn’t had description previously so it would be our first time adding description but you can use this function however you like. This function takes in three parameters, name of the model, version of the model and new description you wanna give to the model version. And it returns a single mlflow.entities.model\_registry.ModelVersion object. Let’s use this function now. Removing the previous code. Client.update\_model\_version, name to be linear-regression-model, version “1” and description which I’m passing pretty much similar to version 2 "First model trained on red wine quality dataset". Let’s run it.



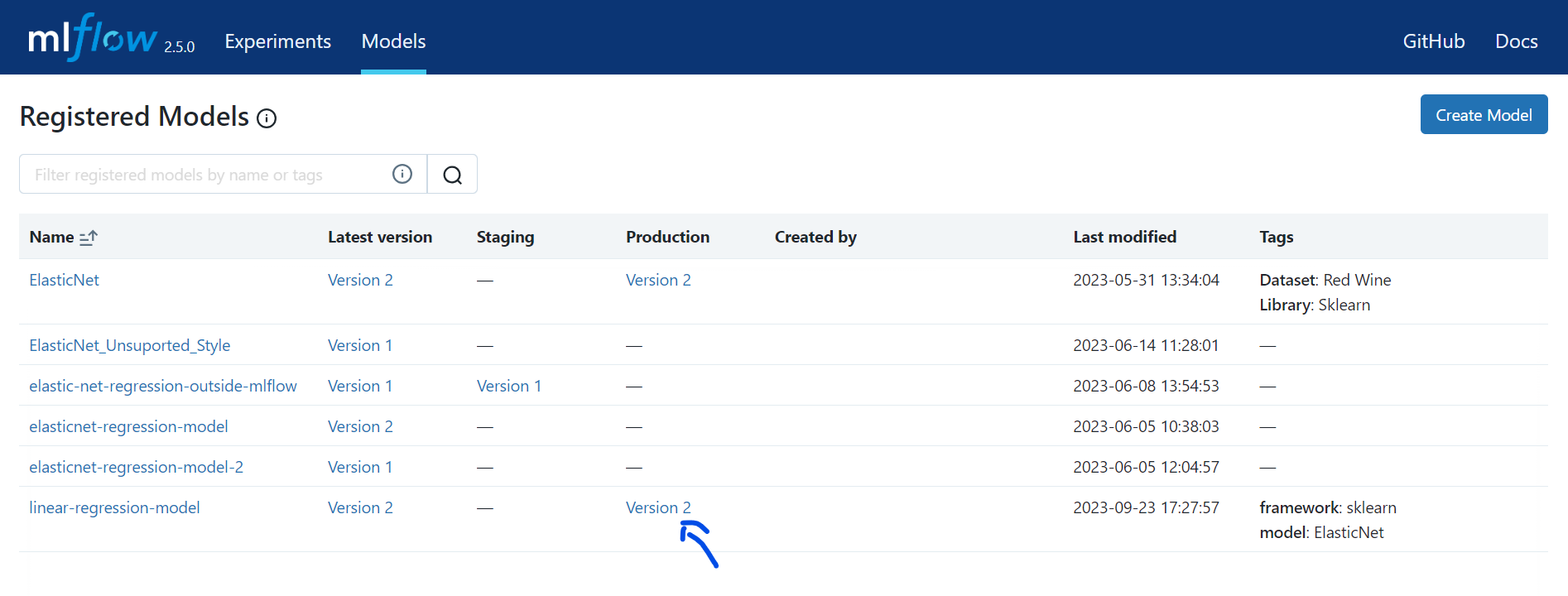
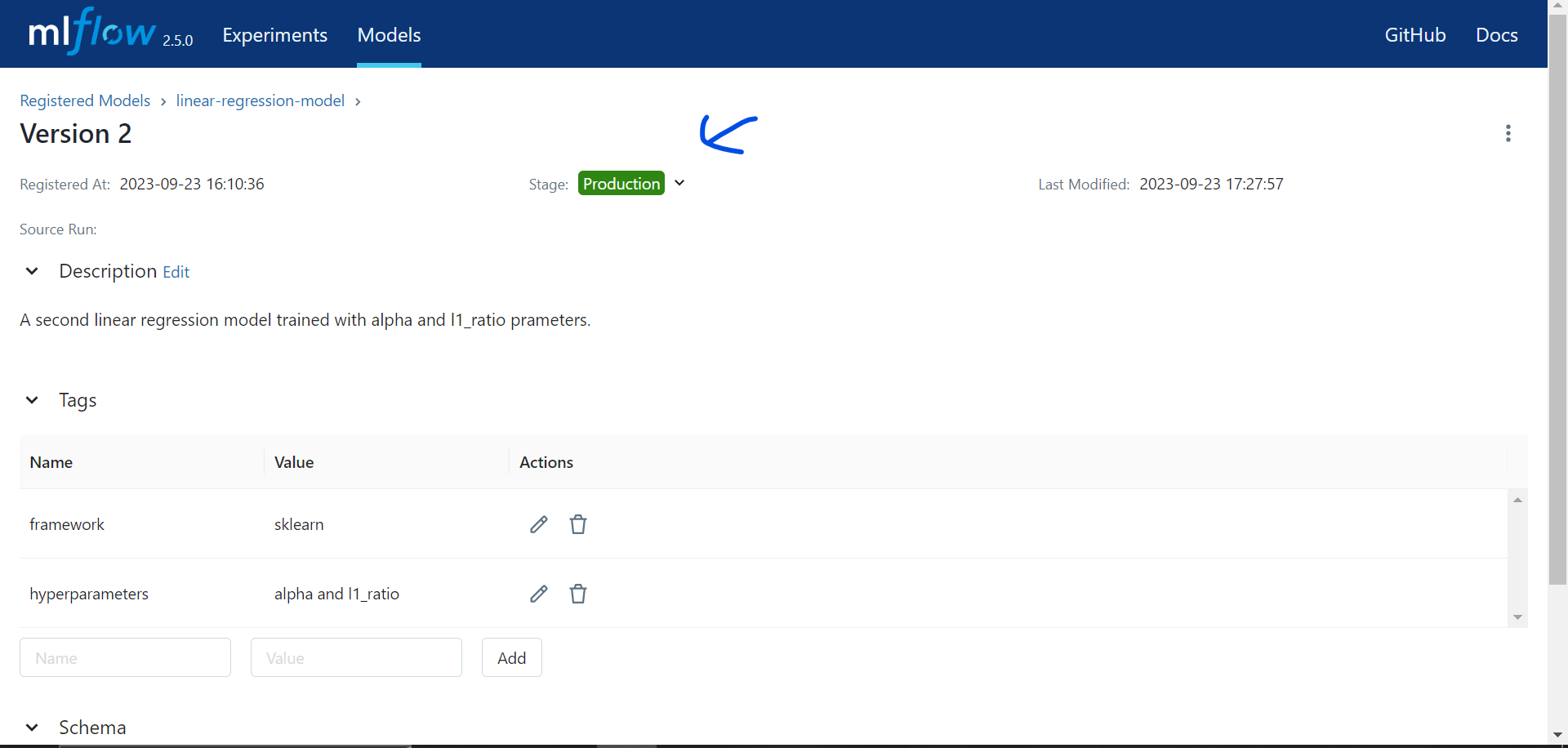
It worked no errors. Let’s also check this on UI. Description has been passed.



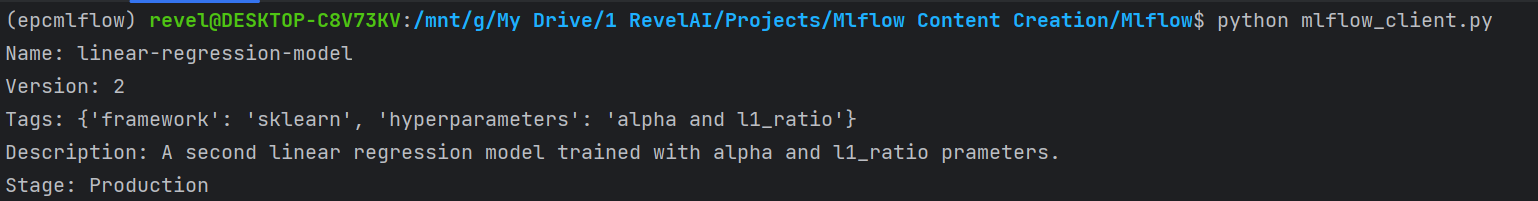
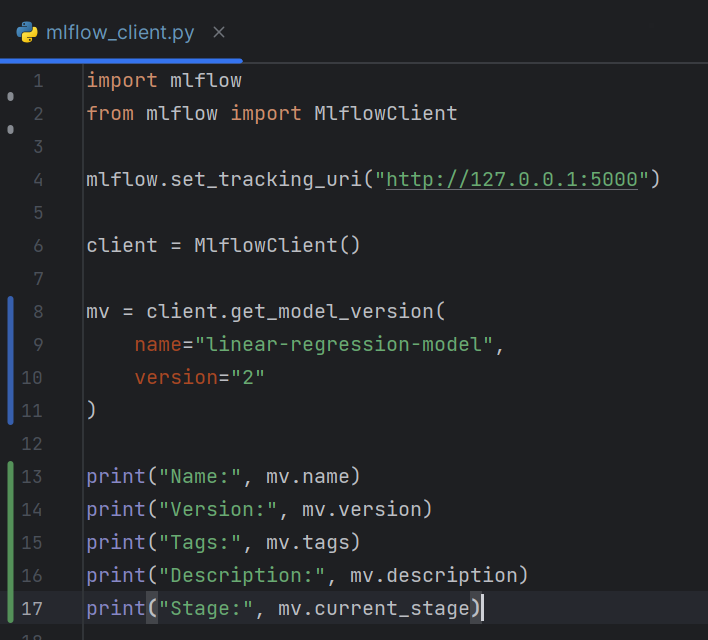
Ok now let’s work with stages. Stages, as a quick recall you set the model to stage, stage means you’re deciding what to do with the model, whether to keep it **None**, keeping it out of concern, to **stagging** which means you’re deciding about it, to **production** which means model is ready and now deployment team can easily put it to production. Lastly, **archived** which simply means we’re done with the model you can delete we it we don’t need it anymore. You don’t jump into delete the model you archive it just in case your mind changes. Let’s transition the model version to some of these stages. The function we use is transition\_model\_version\_stage. It takes four parameters, name of the model, version of the model, stage you wanna give in, lastly the archive\_existing\_versions parameter which if set to True, all existing model versions in the stage will be automatically moved to the “archived” stage. But only valid when stage is "staging" or "production" otherwise an error will be raised. And this function returns a single mlflow.entities.model\_registry.ModelVersion object as most. Now let’s use this function. Removing the previous which we don’t need. Client.transition\_model\_version\_stage, name equals linear-regression-model, version “2”, and let’s stage model version 2 to production. Let deployment take it away from us. And archive\_existing\_versions to False. I don’t wanna lose any of my model versions. Now let’s run the code and set the model version 2 to production.

Well it worked with no errors. Lets also see this on UI. Going to Models. Version 2 is being shown to production column as well. You can look. But still opening the registry, and opening the version 2. The stage has been shown here as well.

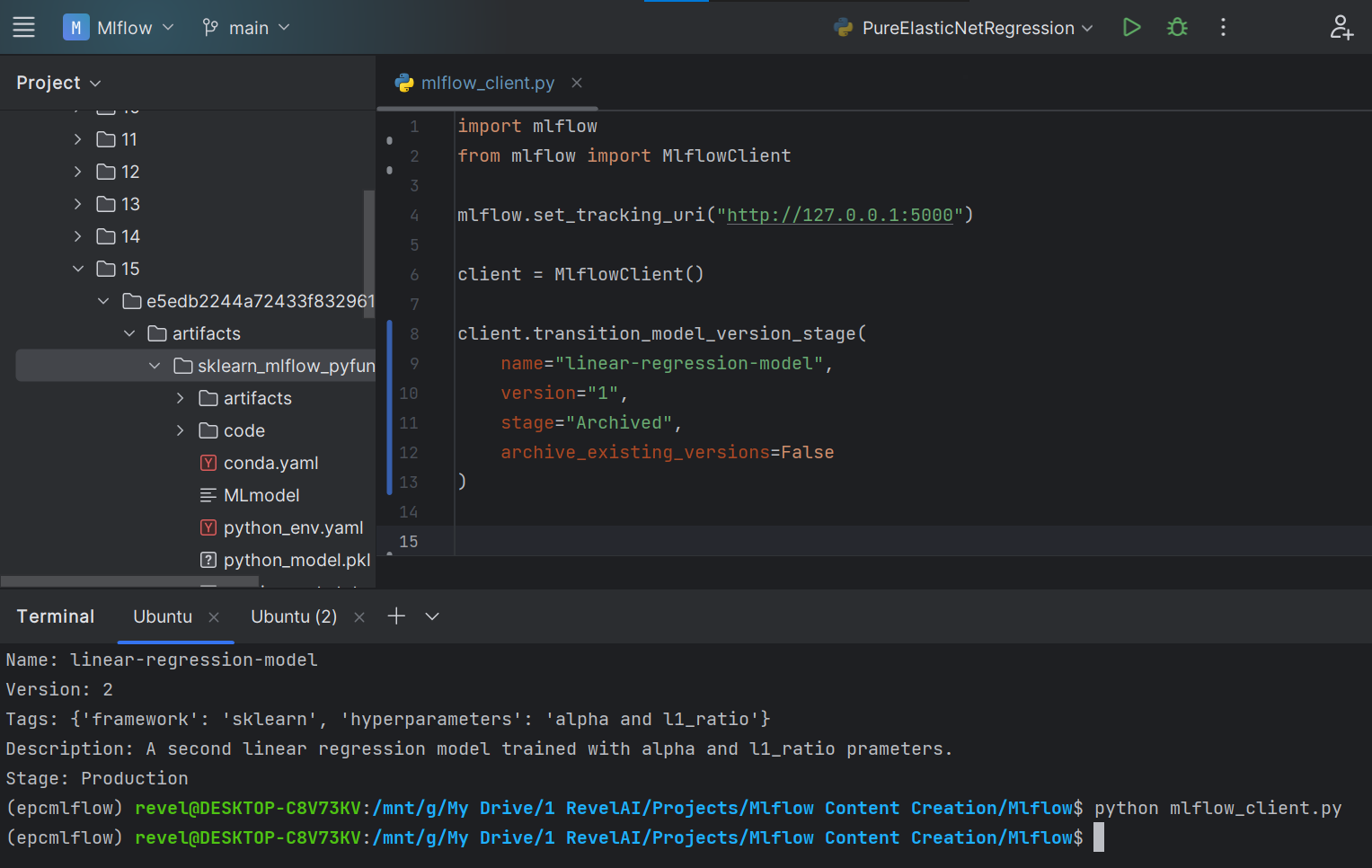
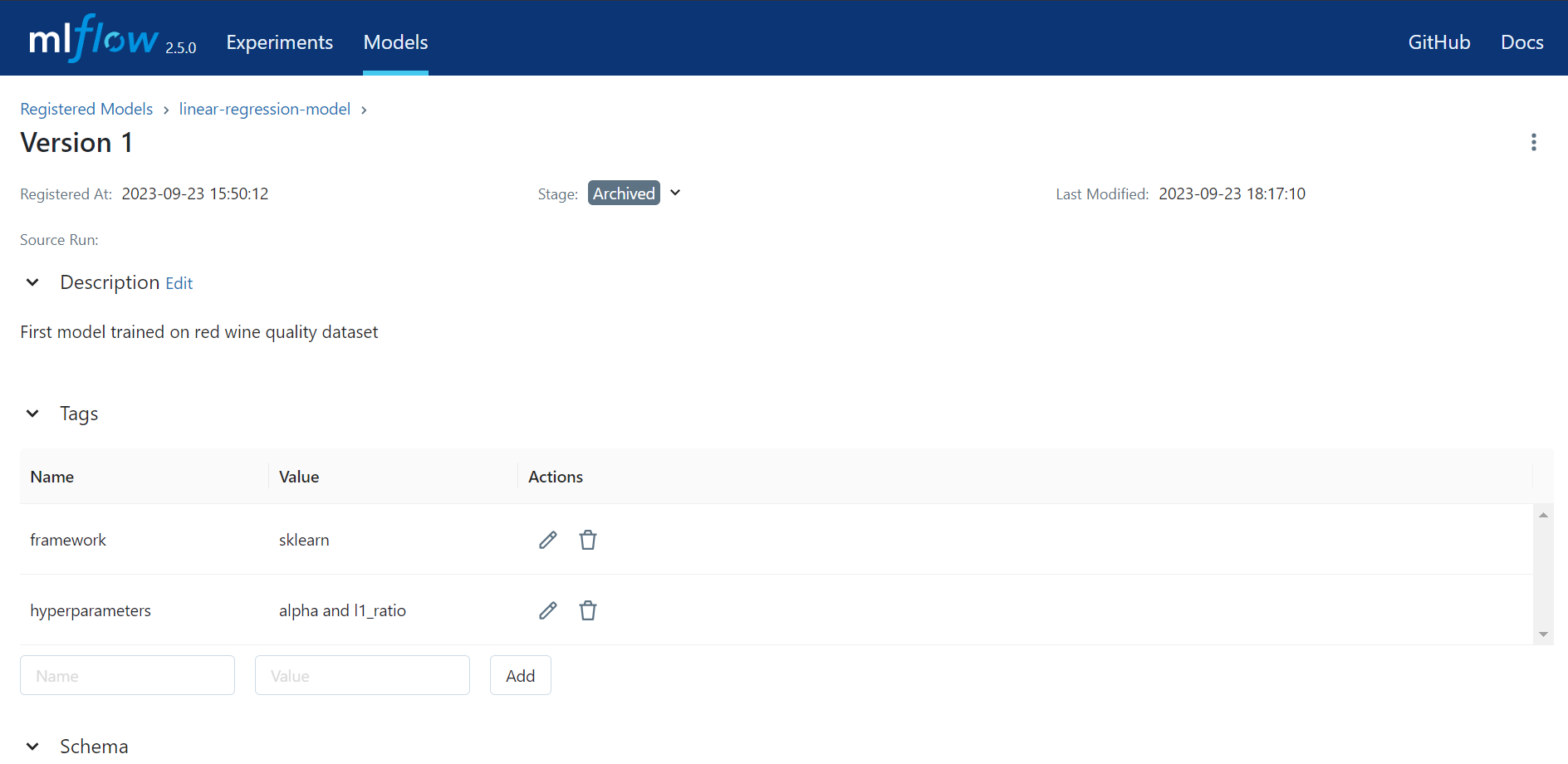
 

Let’s learn how to retrieve model versions. We have three function to retrieve the model get\_latest\_version, get\_model\_version and get\_model\_version\_by\_alias. The get\_latest\_versions is use to get the latest model versions based on the stage. If no stages have been provided it simply returns the latest version of each stage. And get\_model\_version gives a single mlflow.entities.model\_registry.ModelVersion object based on the name and the version of the model. The get\_model\_version\_by\_alias function gives you a single mlflow.entities.model\_registry.ModelVersion object by the name and alias of the model. Now I’m not gonna look into all of them as this section isn’t about model registry but more on trying to teach you mlflow client. So I’m just gonna look into get\_model\_version which seems like middle line between these three. This function takes in name and version of the model as input and returns a single mlflow.entities.model\_registry.ModelVersion object. Now let’s use this function. mv equals client.get\_model\_version, name linear-regression and version 2. The mv object is now the mlflow.entities.model\_registry.ModelVersion object of model version 2. Let’s print out some of the useful info of it. I’m printing out name, version, tags, descriptions and current\_stage. Now let’s run the code. On running you saw it retrieved model version 2 and print out useful insights of it.

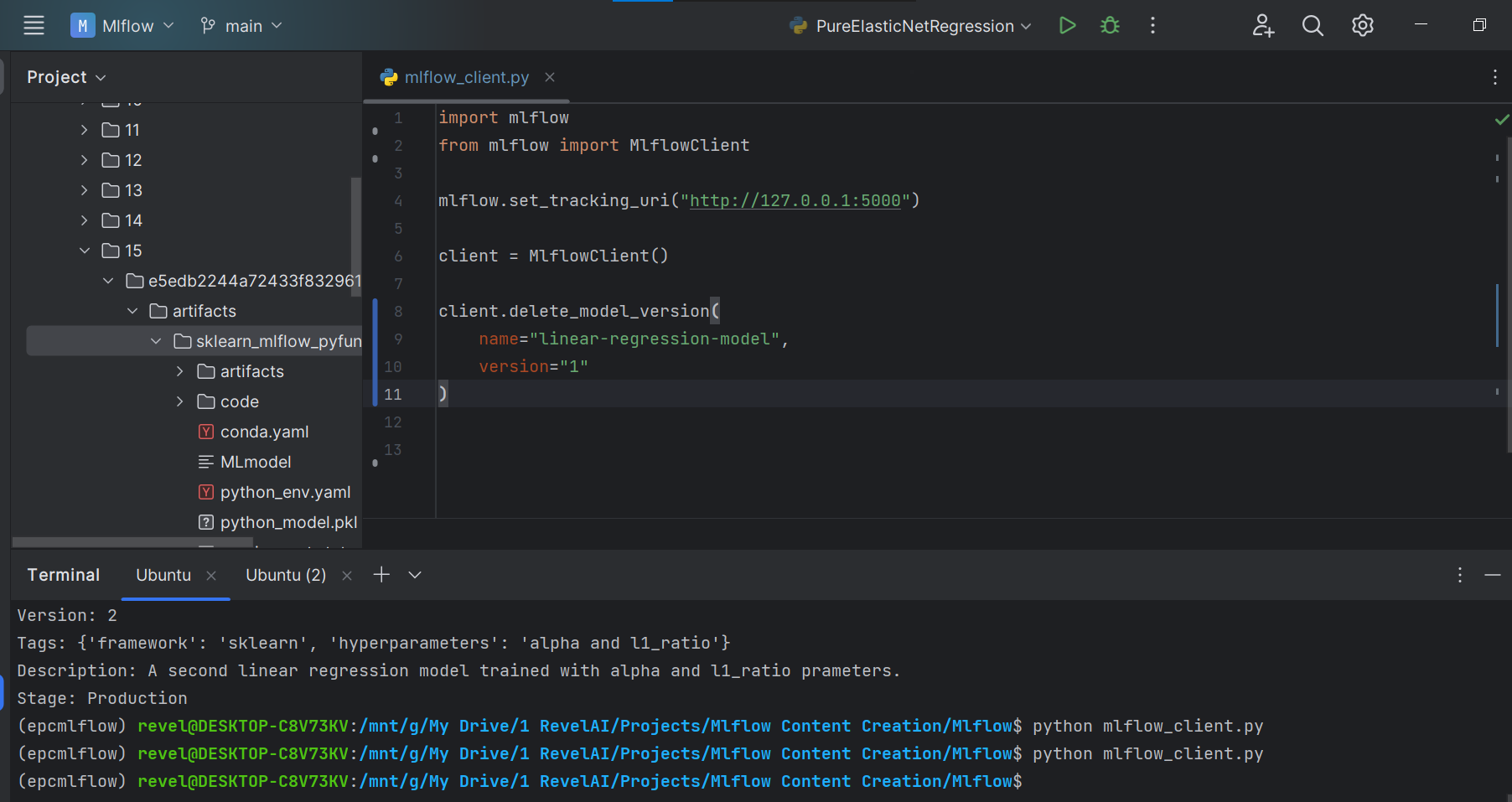


Let’s transition model version 1 stage to archive and delete. We have seen transition\_model\_version\_stage function and now let’s look into delete\_model\_version function which is used to delete model version in the backend. This function takes in name and version of the model as input and delete that model version. Now let’s get the work done. Removing the previous code that we don’t need. Client.transition\_model\_version\_stage, name linear\_regression\_model, version “1”, stage “Archived”, archive\_existing\_versions equals False. Lets run it and check this on UI.

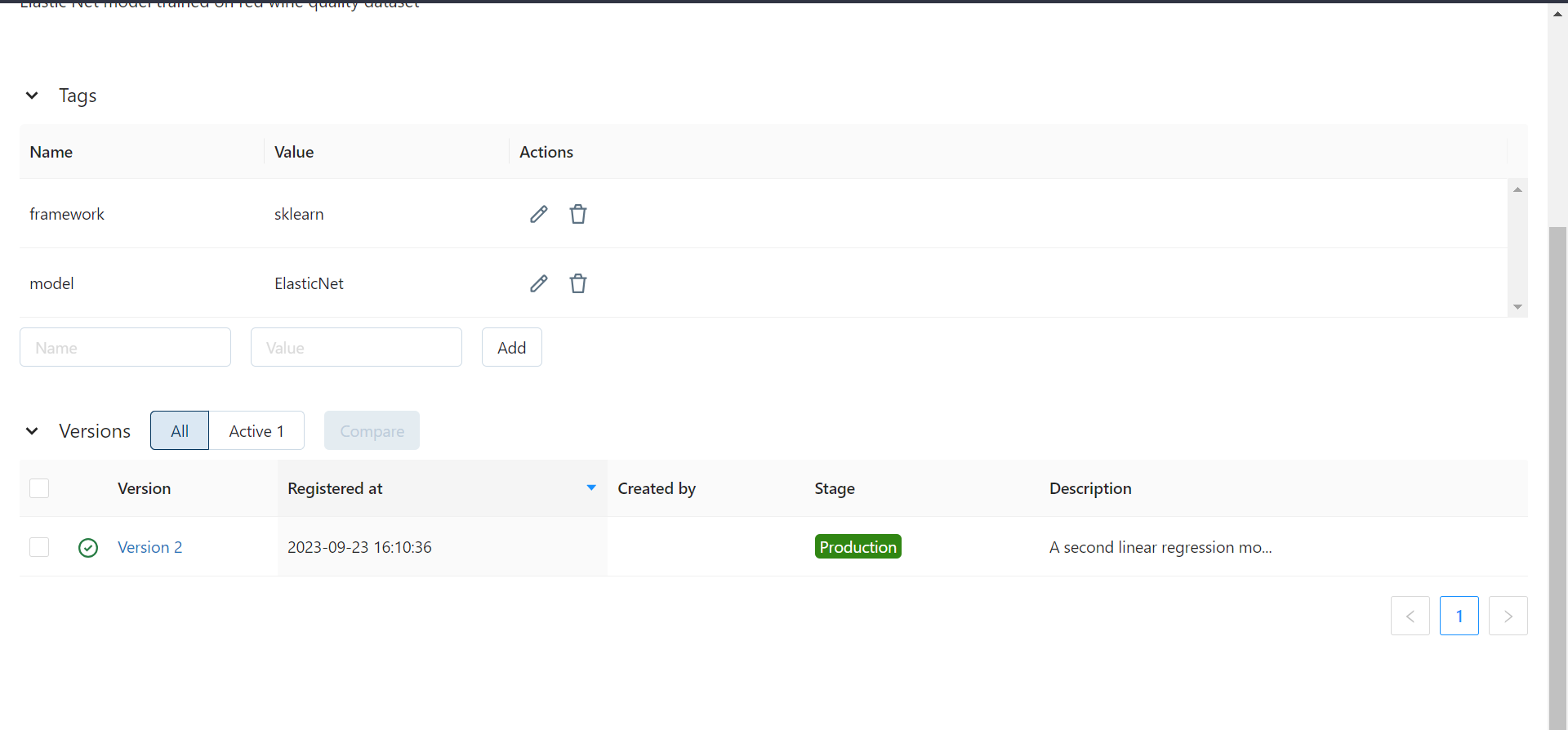
It ran well. Moving to UI, models, then registry, than version 1 and there you go it is set to be archived.

Now let’s delete model version 1. Removing transition\_model\_version\_stage expression. Let’s use delete\_model\_version. Client.delete\_model\_version, name “linear-regression-model” and version “1”. Now let’s run the code and delete model version.



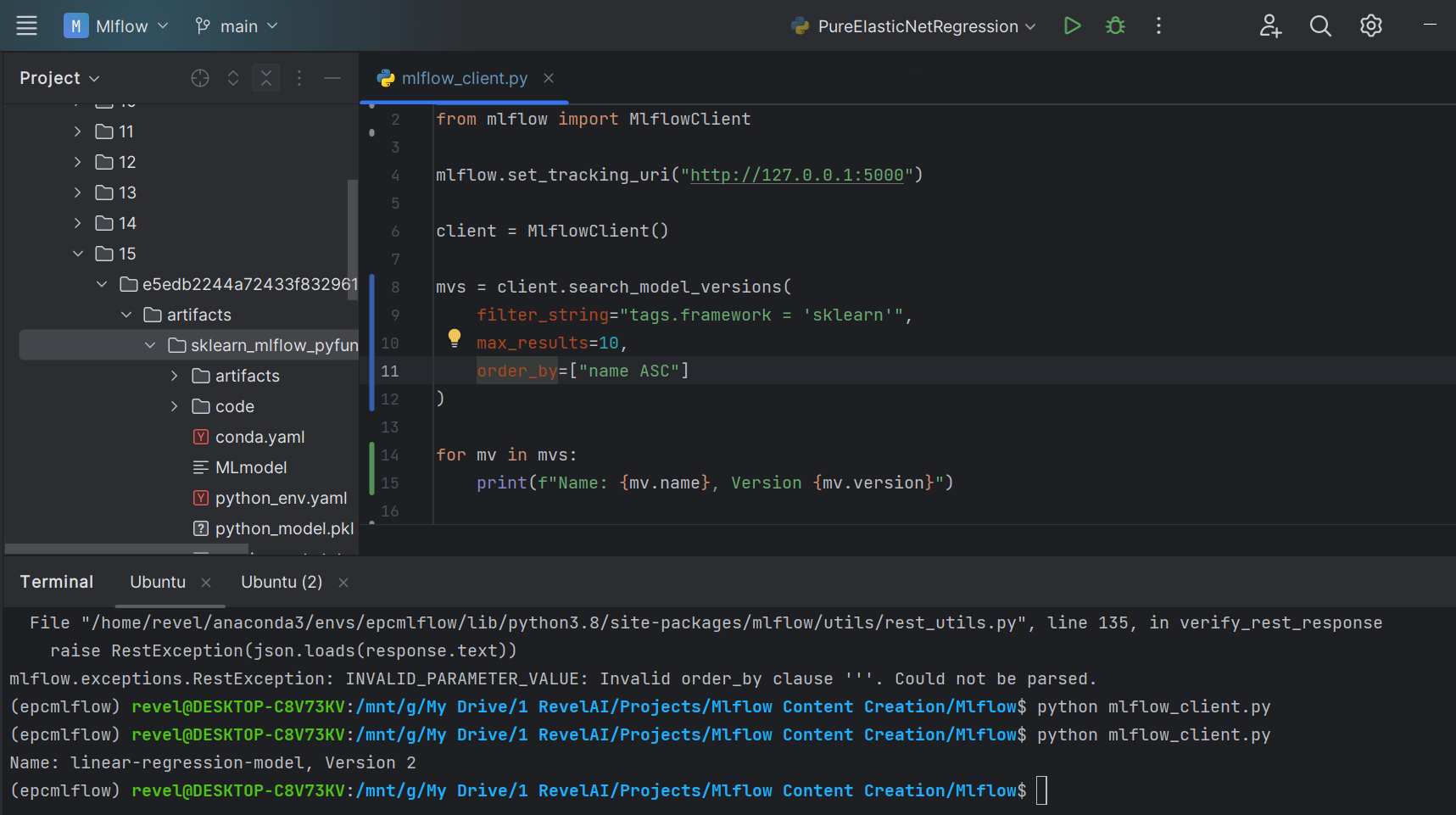
Ok so it ran well. Let’s check this on UI. Moving to Models, Registry, refreshing the page a bit and there you go Model version 1 is no longer there.



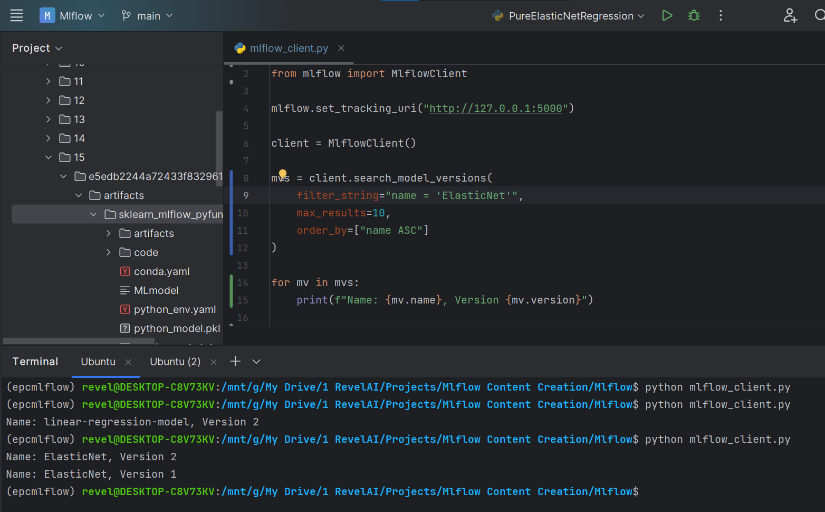
The last function I’m gonna look at is search\_model\_versions. The search\_model\_versions functions is almost same as search\_experiments and search\_runs. The only difference is search\_model\_versions is to search for your desired runs in model registry while search\_experiments and search\_runs are used to search for desired experiments and runs. A little difference here the identifiers in the filter string are name, source\_path the model version source path, run\_id the id of the mlflow run which generated the model version and lastly tags. The rest of the parameters and the whole procedure is same. Let’s look for the model which has tag framework:sklearn.

Removing the previous code that we don’t need. Client.search\_model\_versions, filter\_string equals "tags.framework = 'sklearn'", max\_results=10, order\_by=["name ASC"]. This returns list modelversion objects lets store it in mvs. Now let’s loop through it and print name and version of the each of the modelversion object in the list.

Ok let’s run it. On running you saw it print out the name and version of the suited model.



Now let’s search for the model with name ElasticNet. All the code will remain same let’s change the filter string a bit, name is equals to ElasticNet. Now let’s run the code.

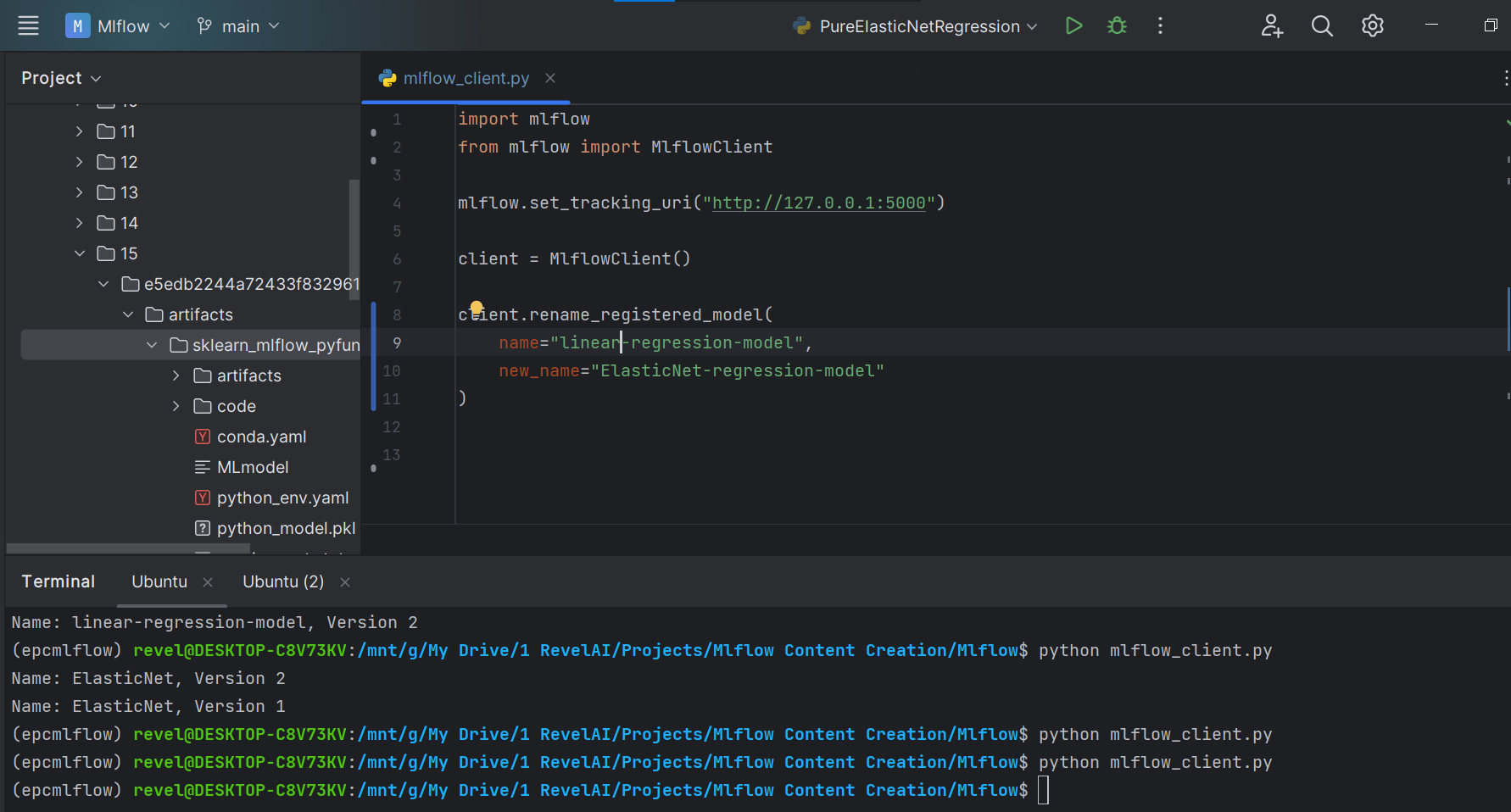


It printed out the two model of same name but two different versions.

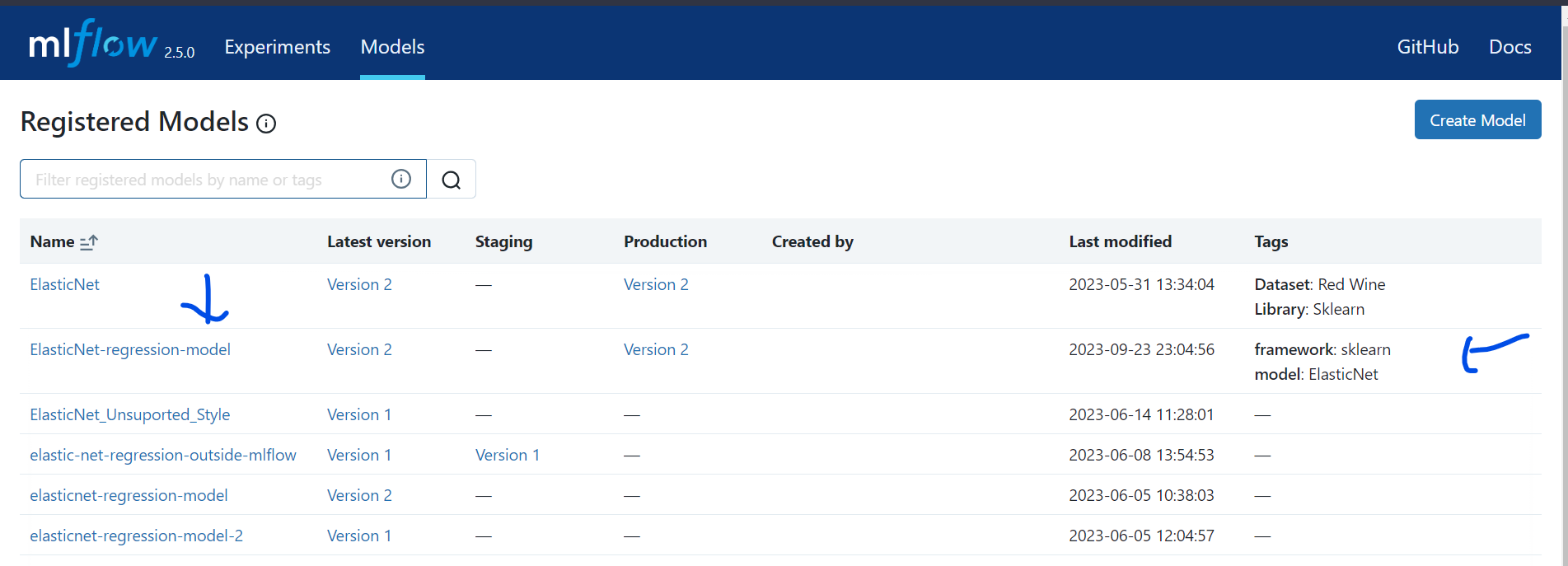
Ok! Now listen there’s nothing I did which you’ve not seen before. Everything is what we have seen before. We’re just doing the same thing but with a different API ok! With that being said lets end our discussion on model versioning managent with mlflow client and kick start model registry management with mlflow client.

Ok So, we’ve already created the registry added two model version to them, played around with them and deleted one as well. Now we’re coming out registry and going to play with registry itself. We have already created one let’s work with that one.

Let’s rename it. To rename we use rename\_registered\_model. This function takes in old register model and new register model as input, and rename the model to the new name provided by you. And also returns a single updated mlflow.entities.model\_registry.RegisteredModel object. Let’s rename our model registry. Removing the previous code. Client.rename\_registered\_model, old name equals linear-regression-model and let’s change it to ElasticNet-regression-model instead of linear-regression-model so new\_name equals ElasticNet-regression-model. Let’s run the code.

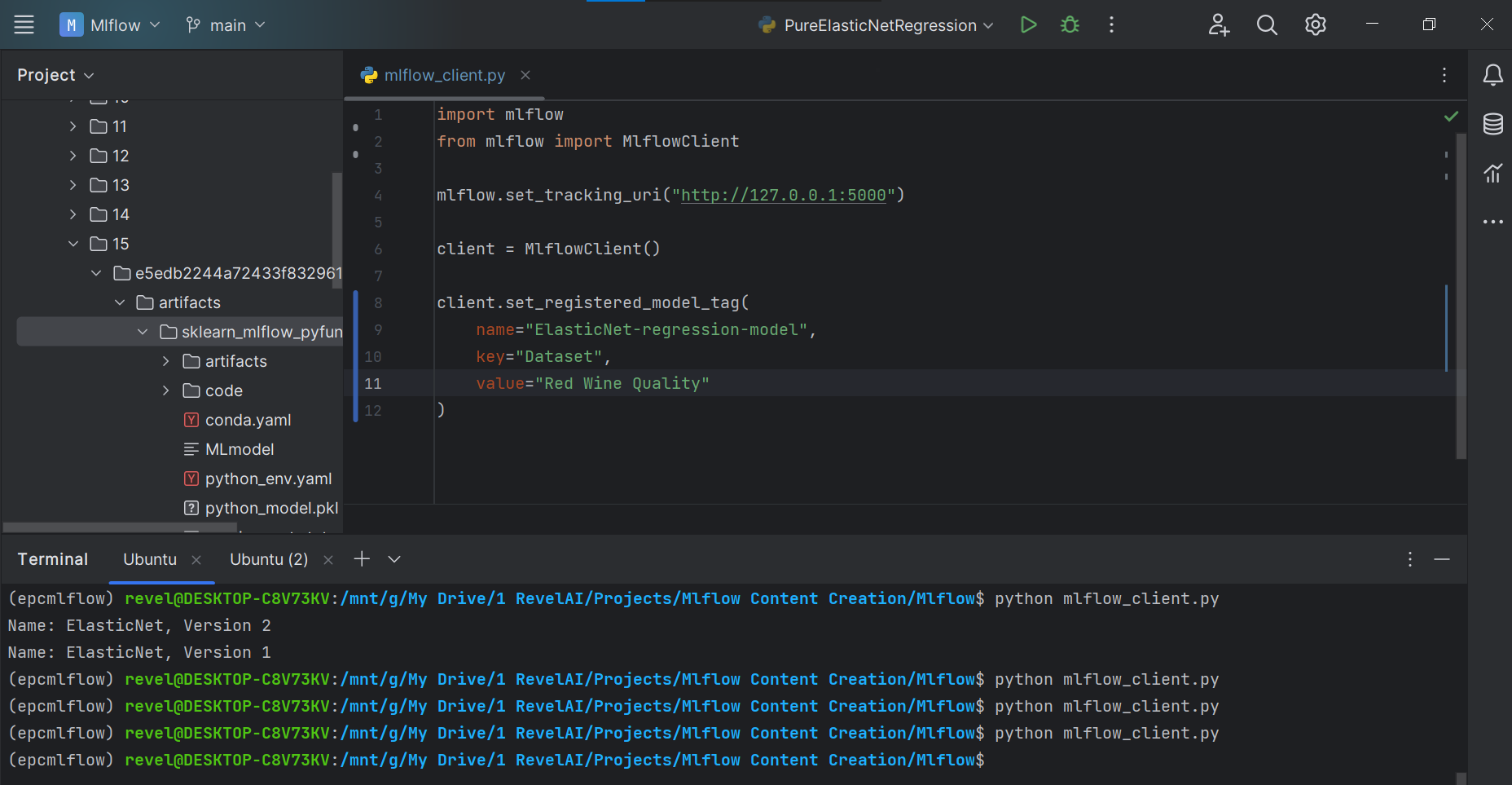
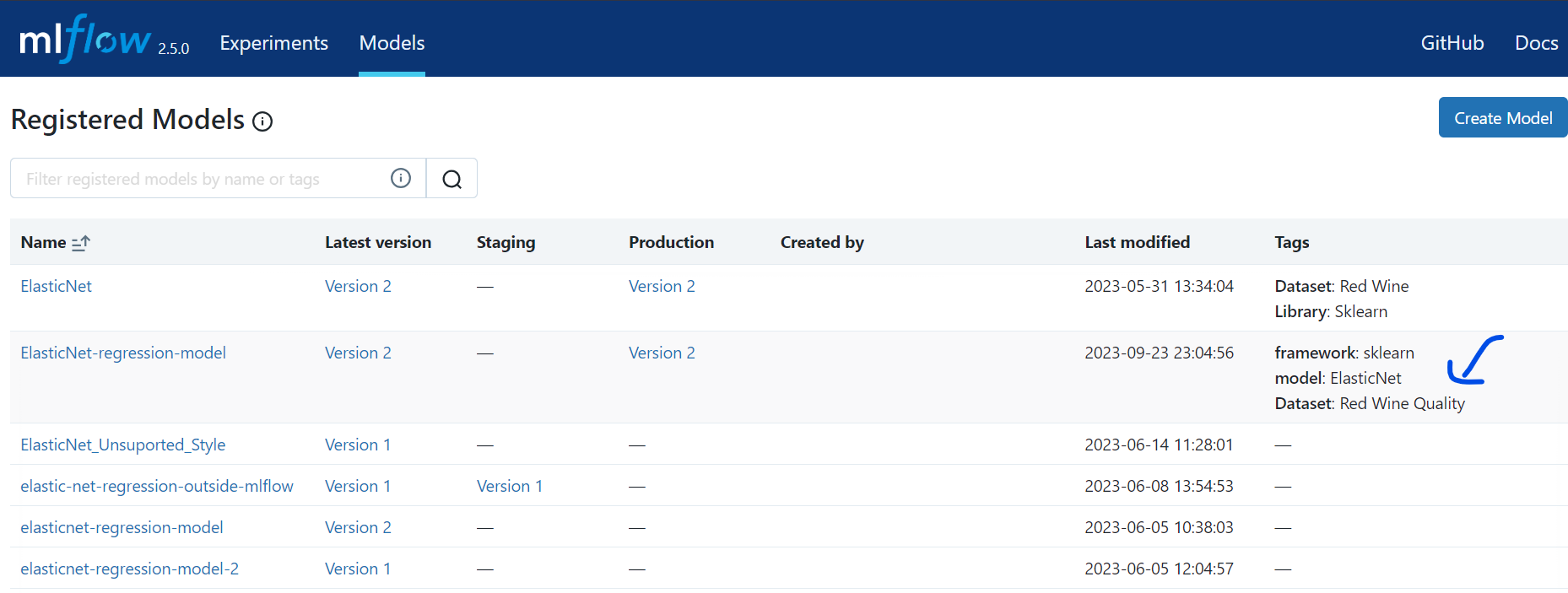


No errors. So let’s move UI, models, refreshing the page. There you go. The new name is there.



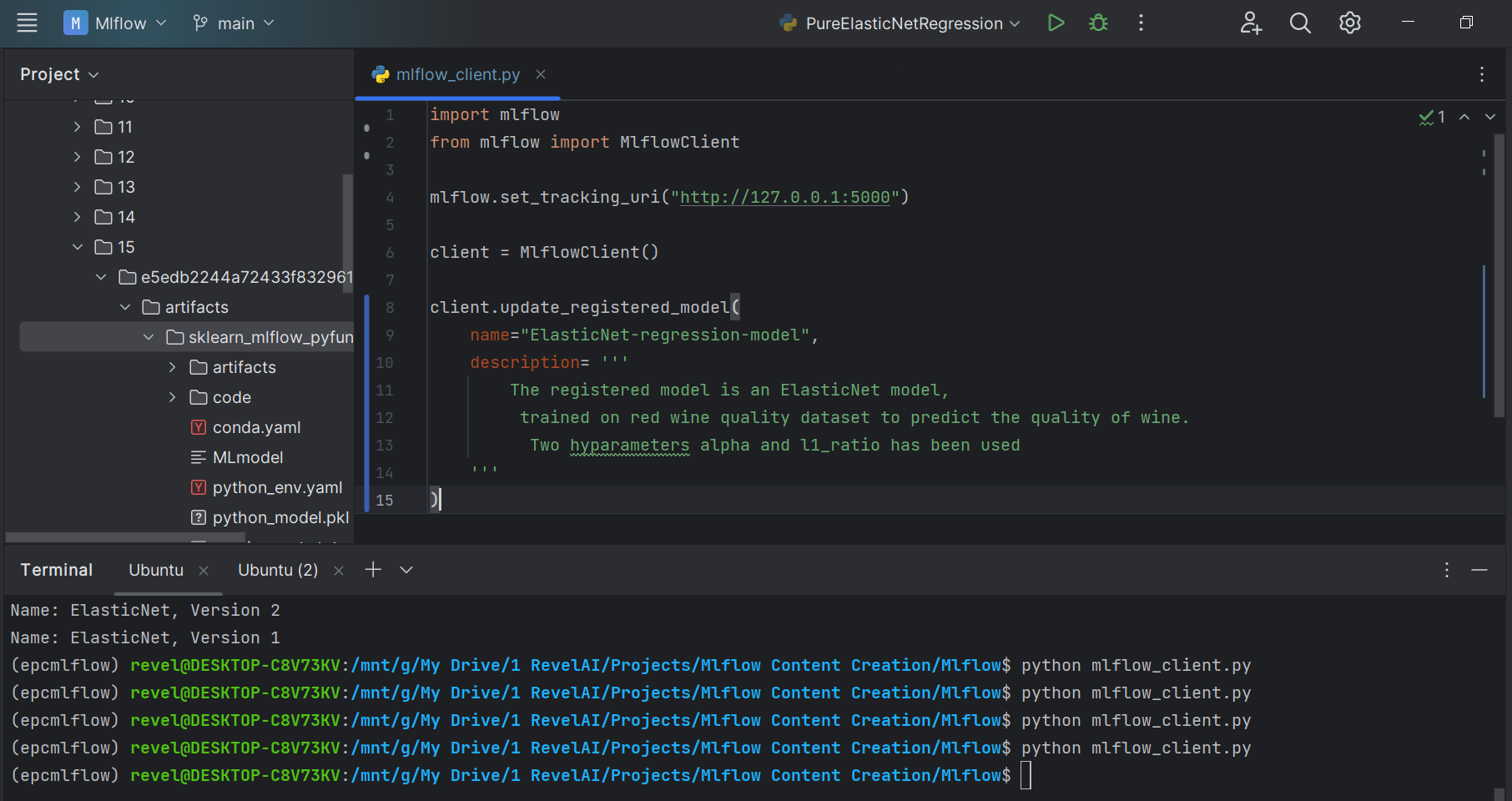
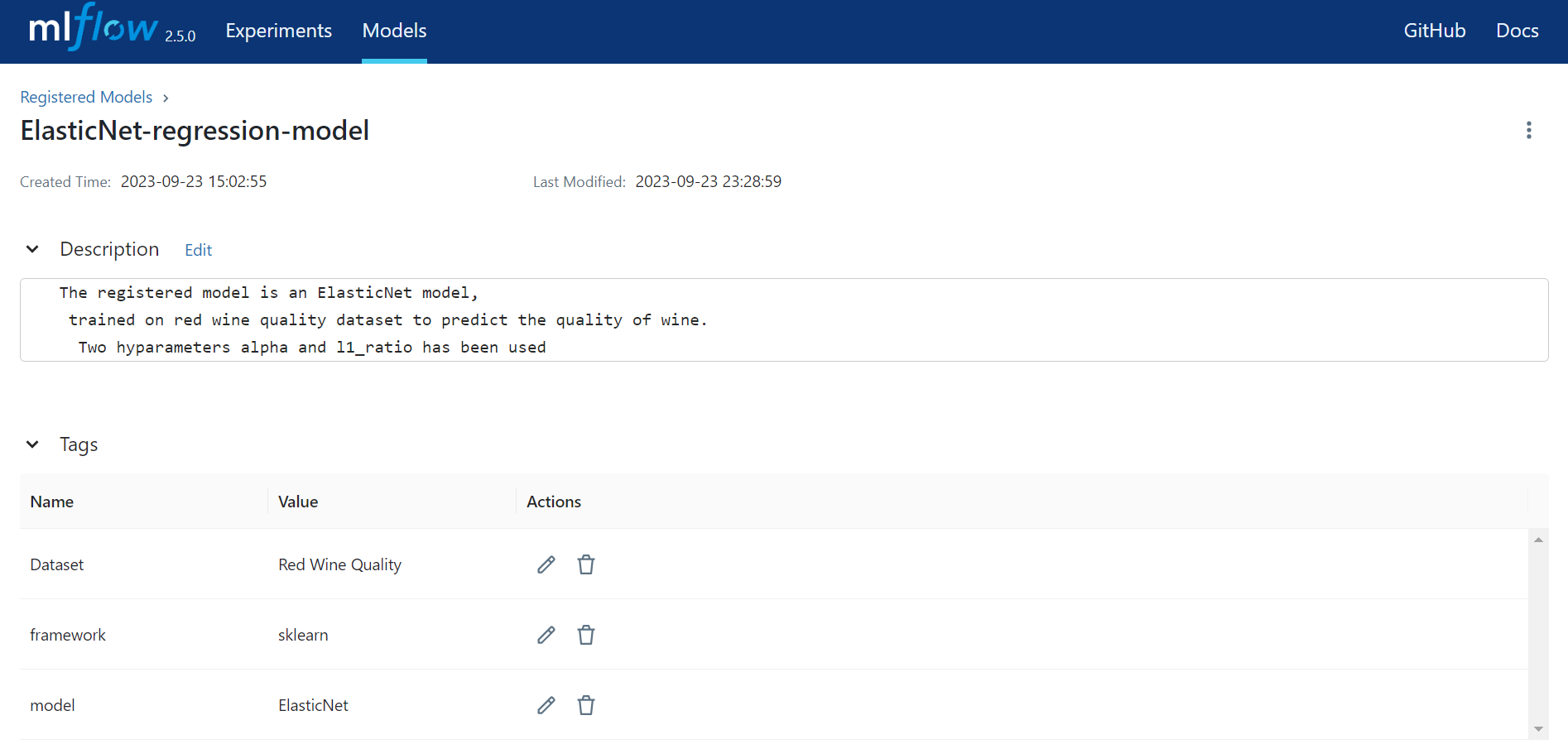
Let’s learn to set a tag to it. I know we’ve already set two tags to it. Let’s set one more. The function set\_registered\_model\_tag is used to set tags to the registered model. It takes name of the registered model, key the tag name and value of the tag as and simply set that tag to the experiment. And doesn’t return anything. Let’s use this in practice. Removing the previous code. Client.set\_registered\_model\_tag, name of the model “ElasticNet-regression-model”, Now let’s just add Dataset equals Red Wine Quality tag to the registered model. So key “Dataset” and value “Red Wine Quality”. Let’s run it.

No errors. Move to UI, to the models, refreshing the page, and there you go, the tag has been set.

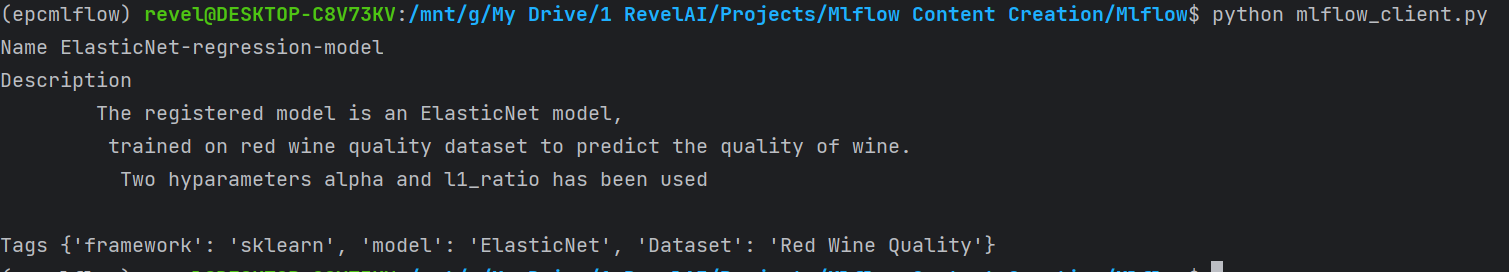
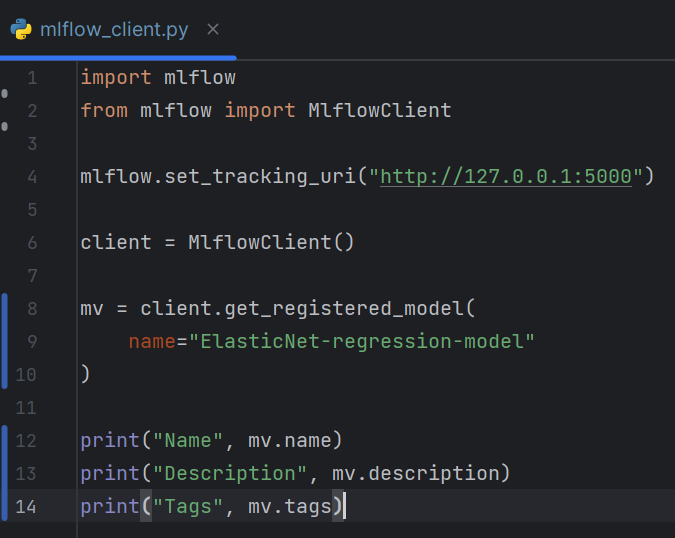
 

Whyn’t update the information. Our previous description is “Elastic Net model trained on red wine quality dataset”. But we’ve n’t talked about hyperparameters here. Whyn’t talk about hyperparameters now. The update\_registered\_model similar to update\_model\_version is used to update metadata of registed model. For now it only updates description. It only takes two inputs name of the model and description, and update the model description. Let’s use it. Removing previous code. Client.update\_registered\_model, name “ElasticNet-regression-model” and description. Let’s add description “The registered model is an ElasticNet model, trained on red wine quality dataset to predict the quality of wine. Two hyparameters alpha and l1\_ratio has been used”. Let’s run the code.

The code ran well. Let’s also see this UI. Moving to UI, to models, getting inside registered model and there you go, new description has been added. It’s a little bit weird because I used triple quote to add description and mlflow seems to pick up the same formatting. Anyways you can use double and single line if you feel uncomfortable with the formatting of the description.

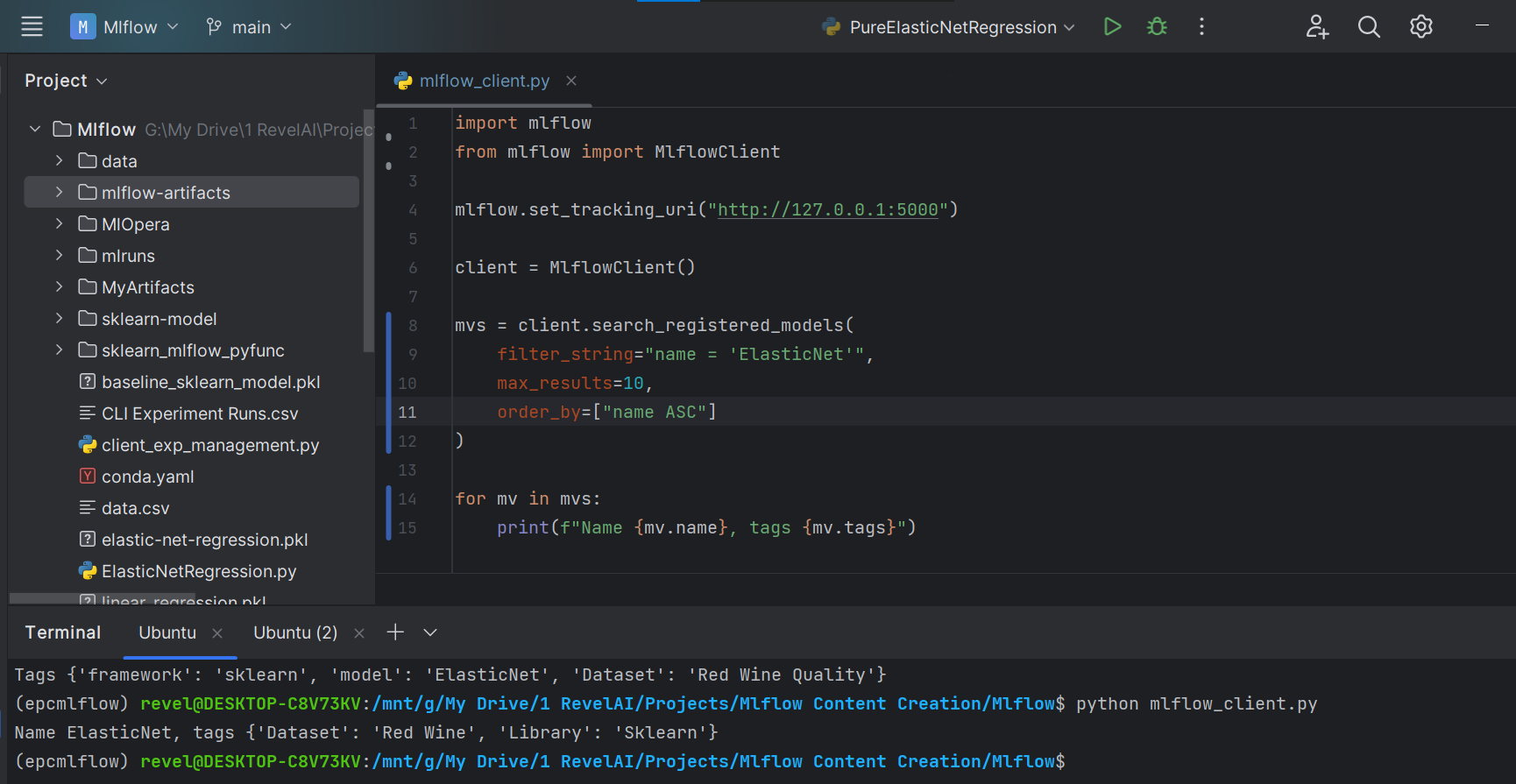
 

Now let’s retrieve the registered model. The function that we have is get\_registered\_model. Its only function available to retrieve the registered model. This function takes in name of the registered model as input and returns mlflow.entities.model\_registry.RegisteredModel object. Let’s use this function, retrieve the registered model from backend and print out some useful insights of the registered model. Removing all the code. mv equals client.get\_registered\_model and name “ElasticNet-regression-model”. Let’s print out the useful insights. I’m printing out name, description, and tags. Let’s run the code

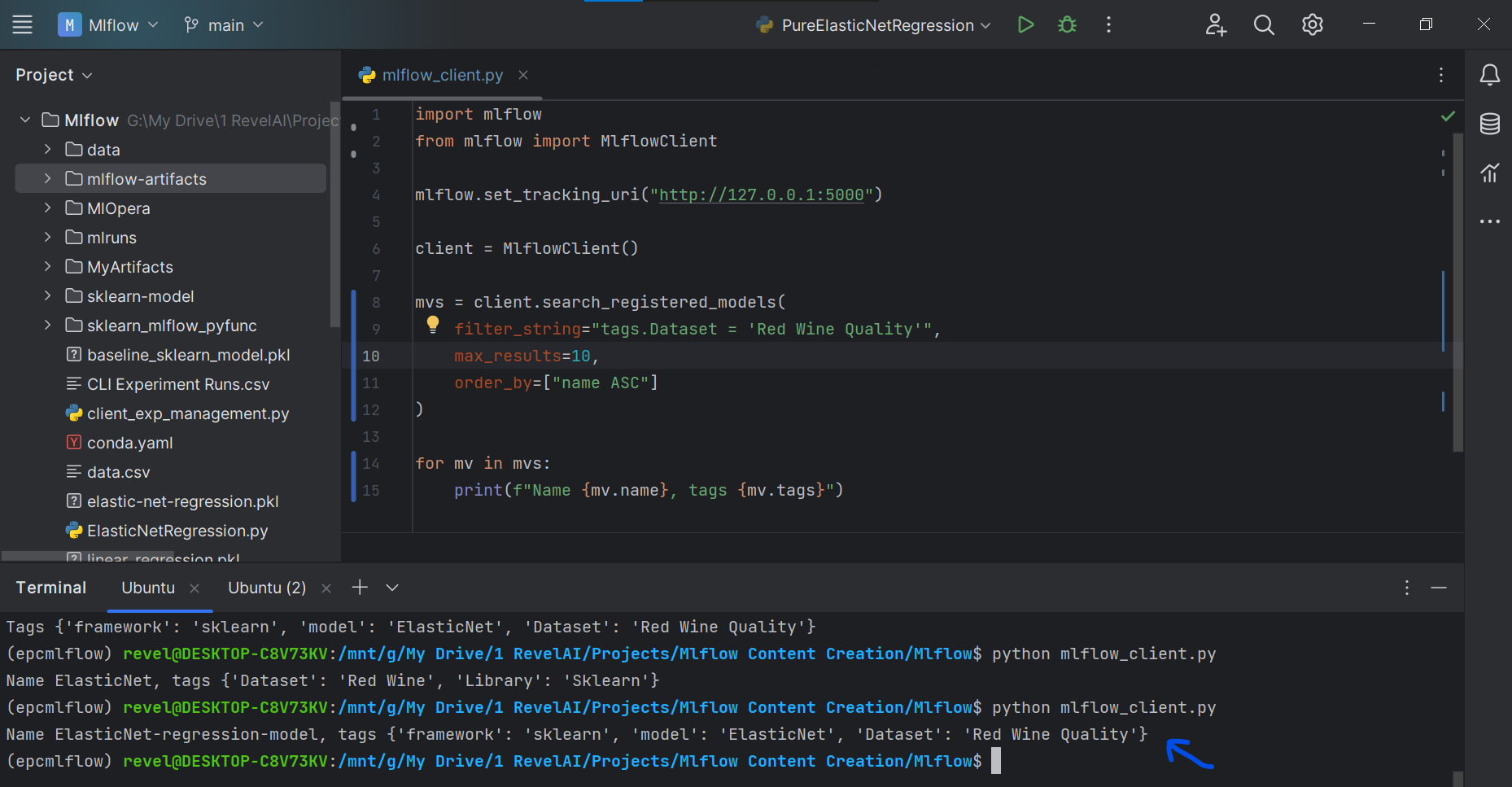


Well it worked well. It retrieved the model and output the useful insights such as name, description and tags of the registered model as I coded.

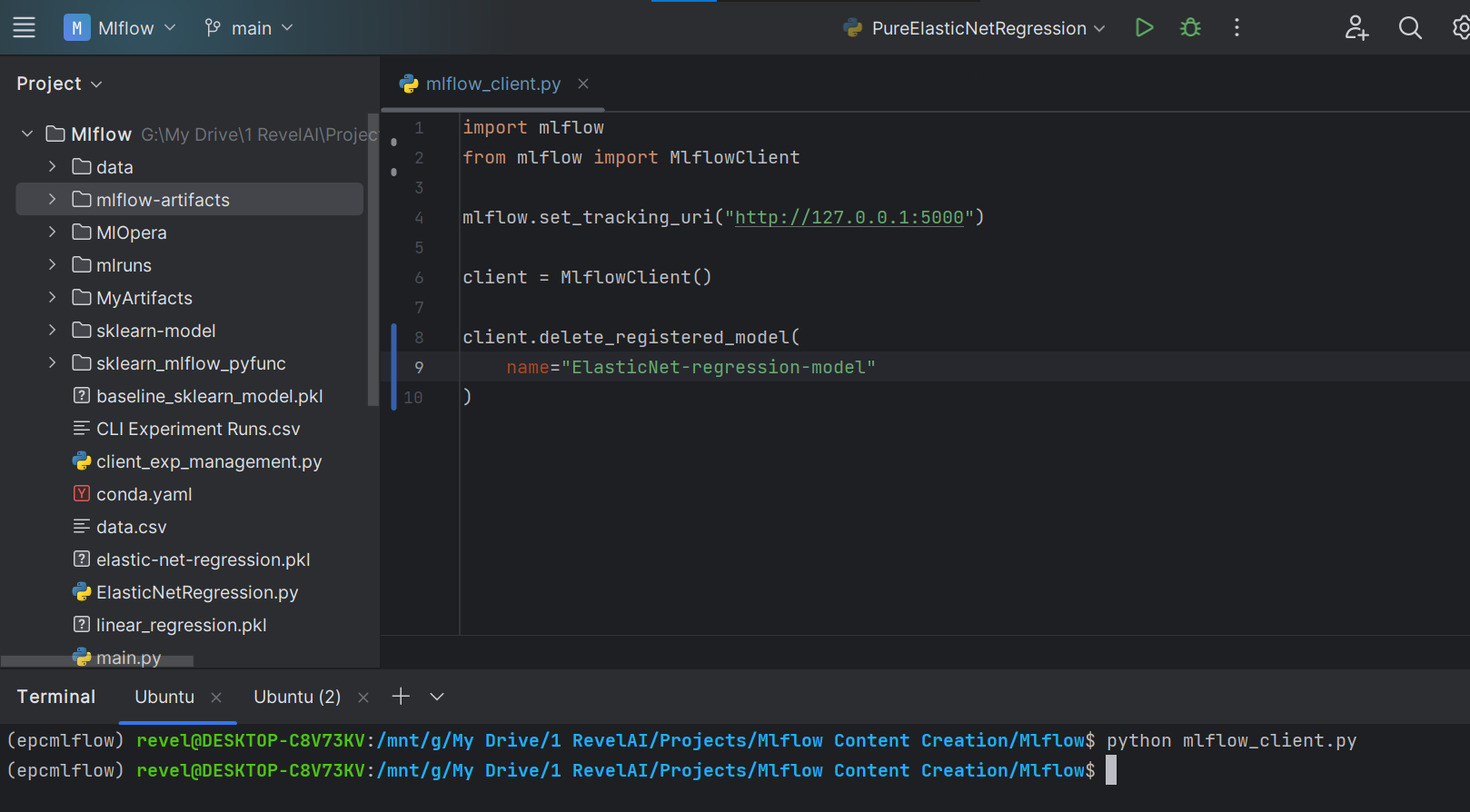
Now let’s discuss one of the type of function which you almost saw in every lecture the searching function. The search\_model\_versions, search\_experiments, search\_runs functions were for searching your suited model versions, experiments, and runs. To search for registed models we’ve search\_registered\_model function. It has the same parameters as others. One change is the identifiers in filter string are name and tags, quite less than others. So let’s use this function. Removing the previous code. mvs equals client.search\_registered\_model, let’s filter the model with name “ElasticNet”. So filter\_string="name = 'ElasticNet'", max\_results=10, and order\_by=["name ASC"], pretty much more like the code of search\_model\_version if you recall. Let’s loop through and print out name and tags of each model. Ok let’s run the code and what it outputs.



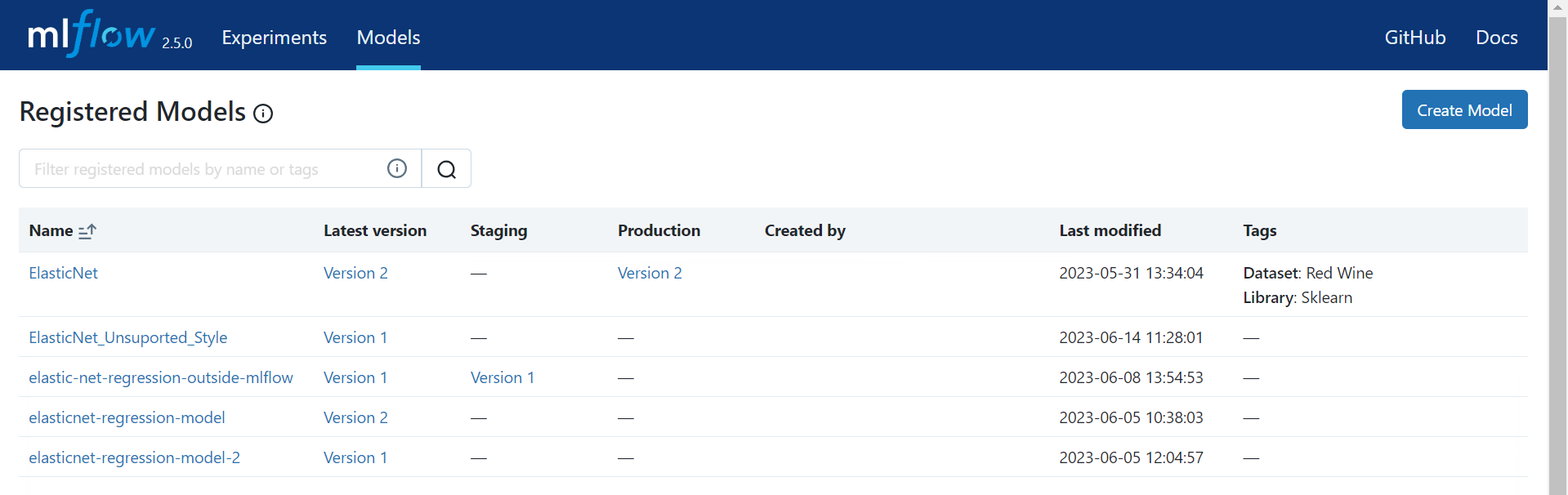
Let’s filter out the registered model with tag dataset equals Red Wine Quality. We don’t have change much of the code just the change the filter string to tags.Dataset = ‘Red Wine Quality’. Ok so lets run the code. You see it outputs the model and tags of the model, which has tag Dataset = ‘Red Wine Quality’.



Well I think that’s all you need to learn about model registry. Let’s work with delete\_register\_model function and end our discussion. This function takes in the name of the model as input and delete the registered model. Let’s use this function. Removing the previous code. client.delete\_registered\_model, name equals “ElasticNet-regression-model”. Let’s run the code.



It ran well that means the registered model has been deleted. Let’s move to UI and check it there. Moving, refreshing, you see the model is no longer there. That means its deleted. Kinda dramatic isn’t it. We started out with registery of that model, we played around with it and at the end we deleted it. Quite dramatic!



Ok! so that concludes our lecture on Model Version and Management. Here we worked with model registry and model versioning with mlflow client. We started out with the creation of empty registery, added model versions to it, played around with them, tagged them and added description to them, searched for the desired, and we did this for both the registery and model versions. Lastly, we deleted model version 1 and at the end we deleted the whole registry that we created. With that being said that’s end of our lecture I will see you in the next lecture.