AI-102 Project

**Problem:**

Select at least 20 images of different automobile types

(e.g., Toyota, Ford, Honda) from various makes and brands.

From these, choose 3 images representing different vehicle types by make and brand. Ensure that the images include both individual vehicle types and mixed vehicle types.

Using Azure AI's computer vision tools, train a machine learning model to classify these different automobile brands and types.

Once the model is trained, test its accuracy and precision by evaluating it with 4 additional images. Finally, provide a detailed report on your findings, including the model's performance and classification results.

Solution Overview :

This solution uses Azure Custom Vision to train a model that can classify different automobile brands. We will select 3 different vehicle types by make and brand, and use at least 20 pictures for each type. We will also test the model with 4 new pictures to evaluate its accuracy and precision.

Step 1: Collect and Prepare Data

* Collect at least 20 pictures for each of the 3 vehicle types (e.g., Toyota, Ford, Honda).
* Make sure the pictures are in JPEG, PNG, GIF, BMP, WEBP, ICO, TIFF, or MPO format.
* Ensure the file size of each image is less than 20MB and the dimensions are greater than 50x50 pixels and less than 16,000x16,000 pixels.
* Create a CSV file to store the image file names and their corresponding labels (e.g., Toyota, Ford, Honda).

Step 2: Create a Custom Vision Project

* Go to the Custom Vision portal and sign in with your Azure account.
* Create a new project and select "Classification" as the project type.
* Choose a domain that is suitable for your use case (e.g., "Generic").
* Upload your images and create tags for each vehicle type.

Step 3: Train the Model

* Train the model using the uploaded images and tags.
* Monitor the training process and adjust the model as needed.

Step 4: Test the Model

* Test the model with 4 new pictures that were not used during training.
* Evaluate the model's accuracy and precision.

Step 5: Deploy the Model

* Deploy the trained model to a cloud or edge device.
* Use the model to classify new images.
* Click to add a cell.

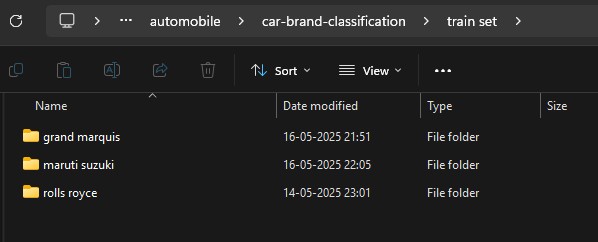
**Personal Solution**

I personally have chosen 3 different car brands

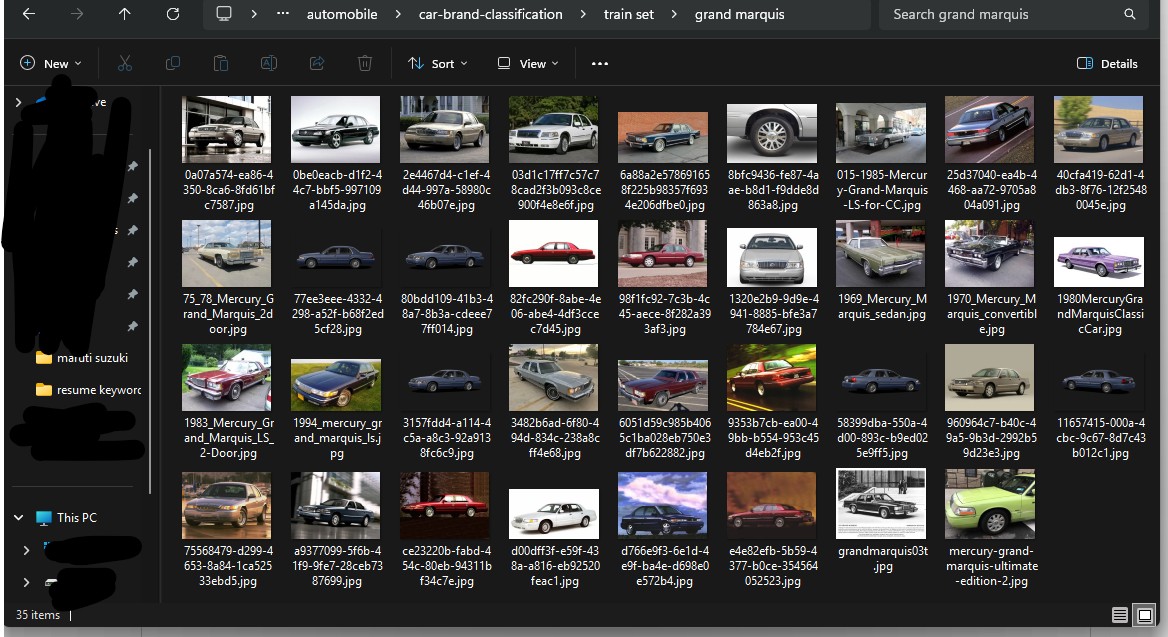
1. Mercury Grand Marquis
2. Maruti Suzuki
3. Rolls Royce

I have data scrapped 35 images from each of this car brands

Then inside the folder “train set” I created separate folder for each car brand.

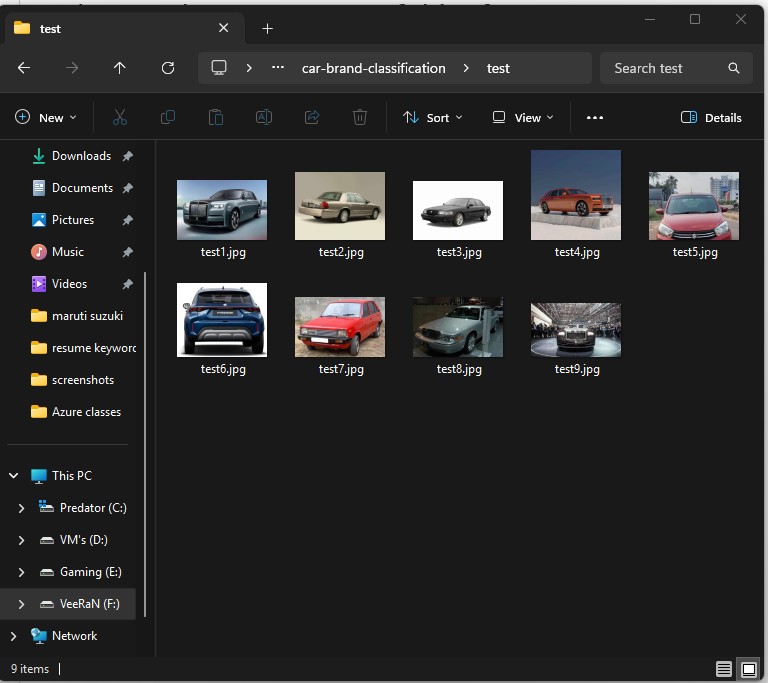


Since each of these folders contain 35 images, each.



The above images represents the 35 different images of the car grand marquis I have done the same for the Maruti Suzuki and rolls Royce as well.

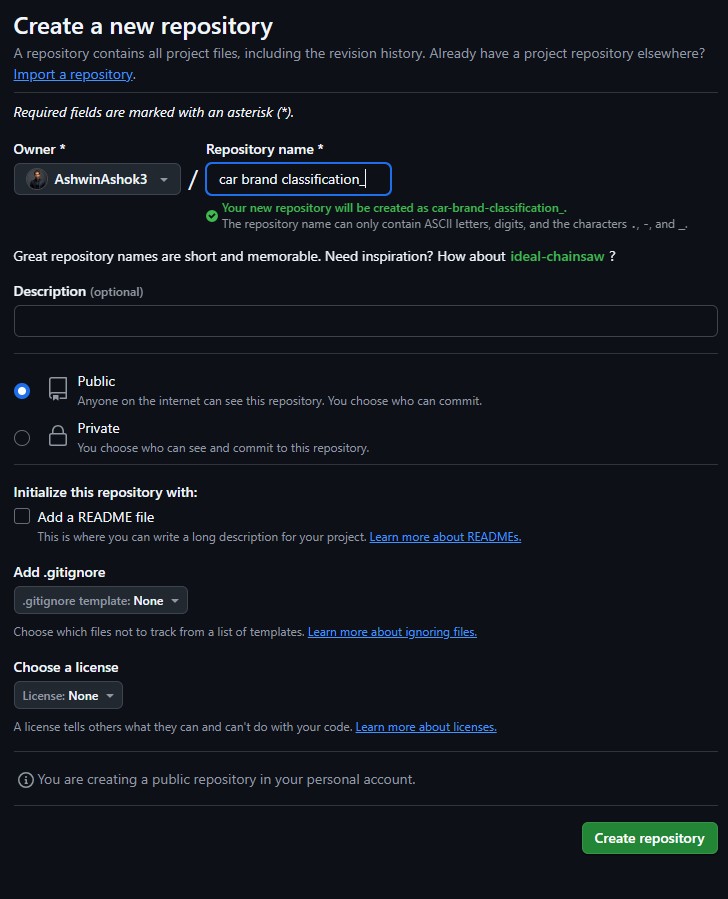
Now next we create a test set of each of these cars though we know what these brands are the custom vision trained model should also know the same so for testing and prediction we will have to keep a separate folder for our test set.



As you can see I have taken 3 images for each car brand and have applied the same and named as test\_num.jpg.

Another important factor to remember is that I have all my images in jpg format so in that way my images will have no problem uploading into custom vision.

Next step I did was to create a repository in github named “car brand classification”.



Now when you arrive at this page name your repo: “repository name”

If you want to add readme file you can do that but I was planning on doing that after I finish this project.

Also we will have to add “.gitignore” file other while running azure SDK’s it will show named too long can’t push or some error like that.

So for now we will leave it as it is and later add them so we will click on create repository.

My next step is to open My IDE in my case it is PyCharm and clone the repository into the folder where train and test set is included.

And then initialized git on to the cloned folder and then connect it to github using tokens and API keys.



Now type git init



Now we will have to connect it to the github apikey

If written properly likewise :

“ git remote set-url origin [https://<api\_key\_or\_token\_key>@github.com/<username>/<repository\_name](https://%3capi_key_or_token_key%3e@github.com/%3cusername%3e/%3crepository_name)> ”

Then now we will move the train set and test folder into this directory and upload them into github using the command line.

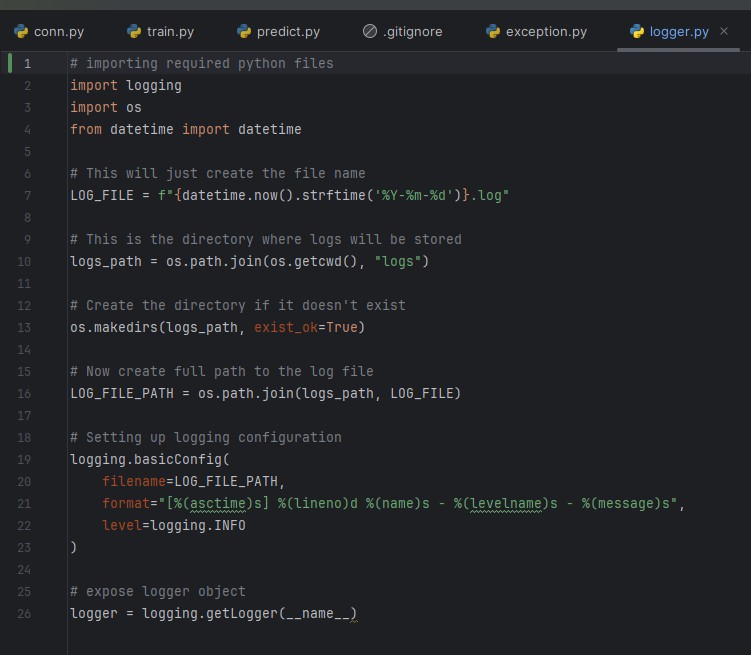
Commands for uploading to push items to github repo for which you have cloned.

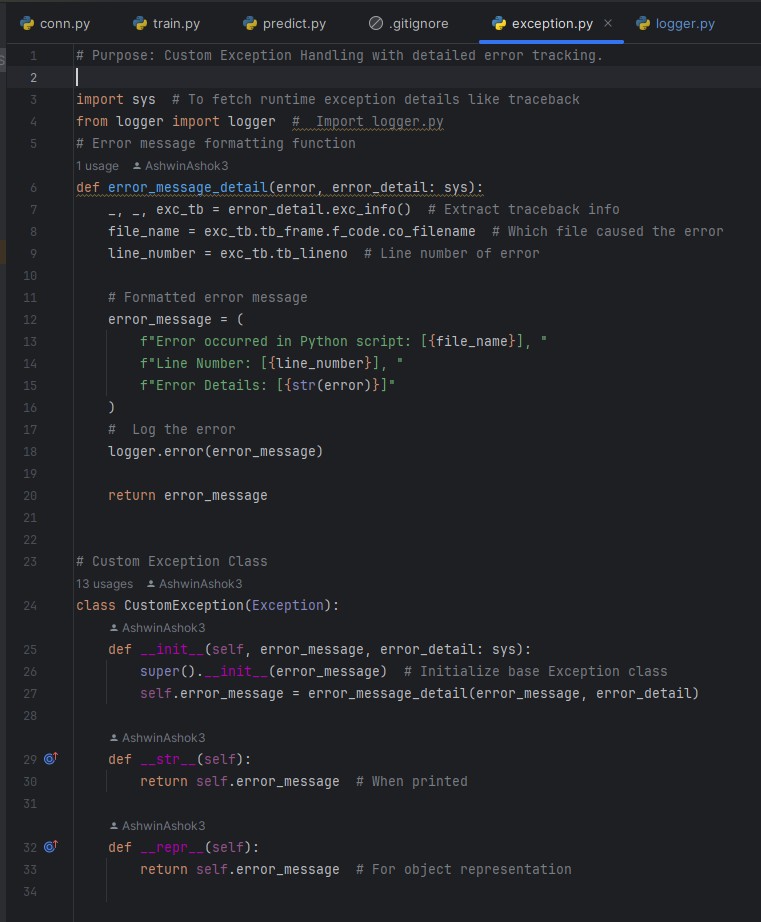
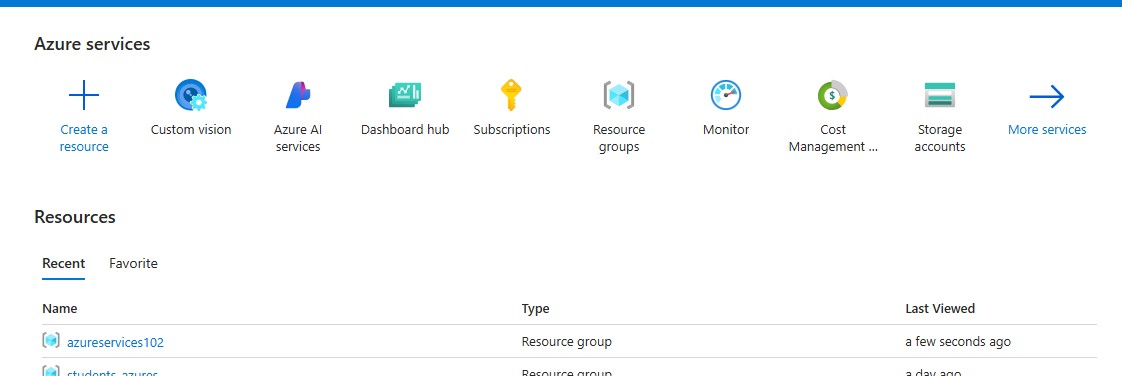
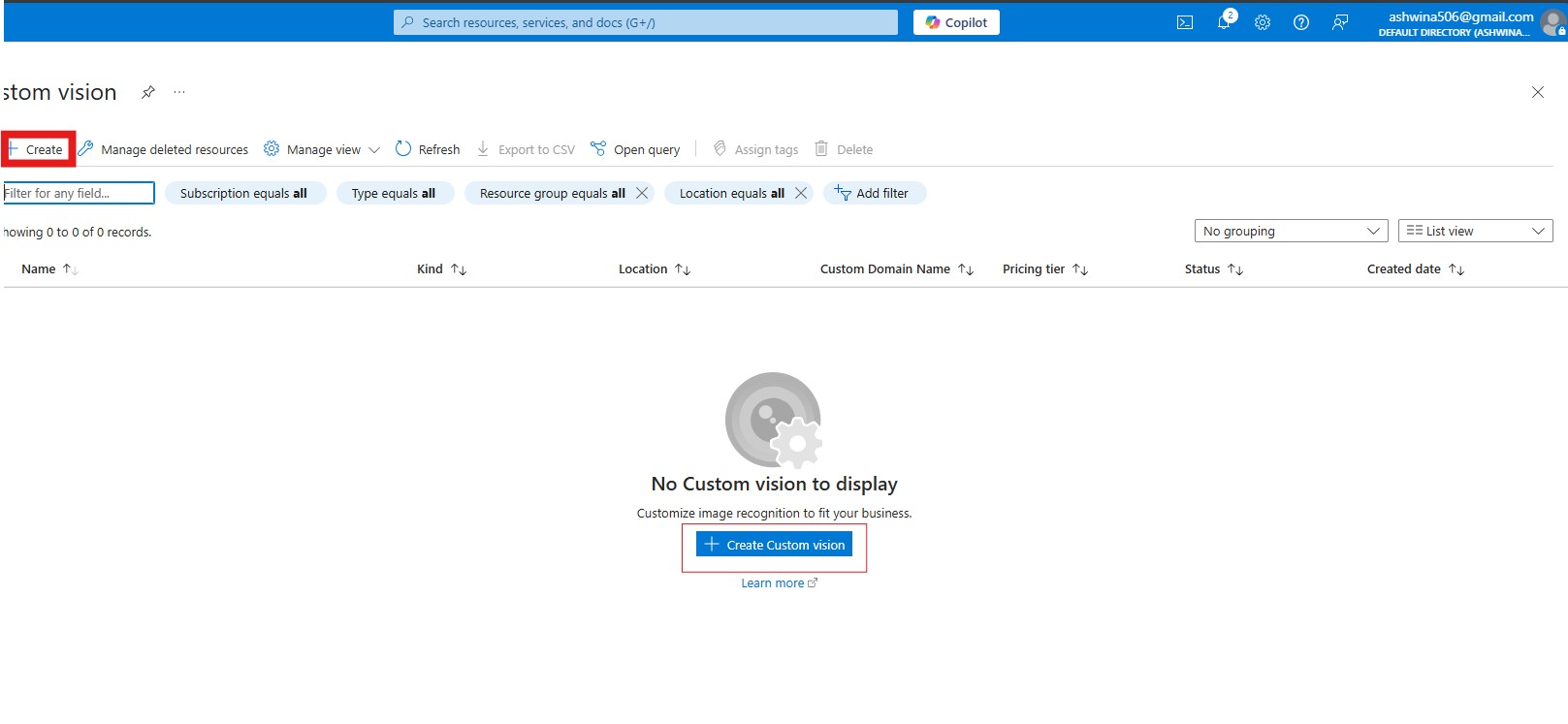
1. git add .
2. git status
3. git commit -m “description for the commit to happen”
4. git push -u origin main

Now that we have uploaded and maintained the same phase as our folder to our github repo.

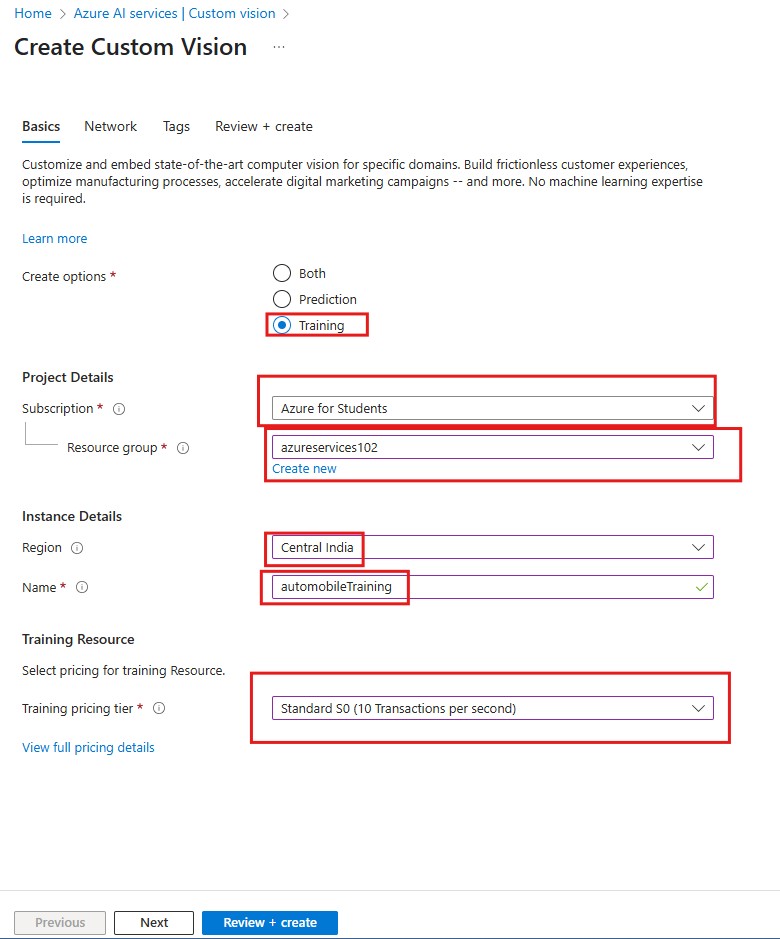
Now we will first install required libraries and environments:

1. Install python ( “python -m venv venv” )
2. Now we will have to install azure services
   1. ( “pip install azure-cognitiveservices-vision-customvision” )
3. Now we will setup our exception handling code
4. But we will have to activate our venv
   1. (“.\venv\Scripts\Activate.ps1”)
5. Now we prepare our logger.py file:

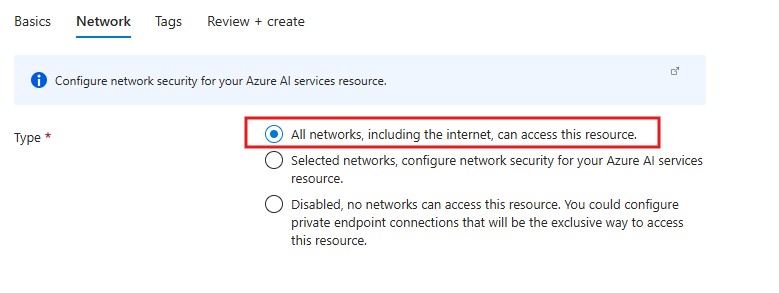


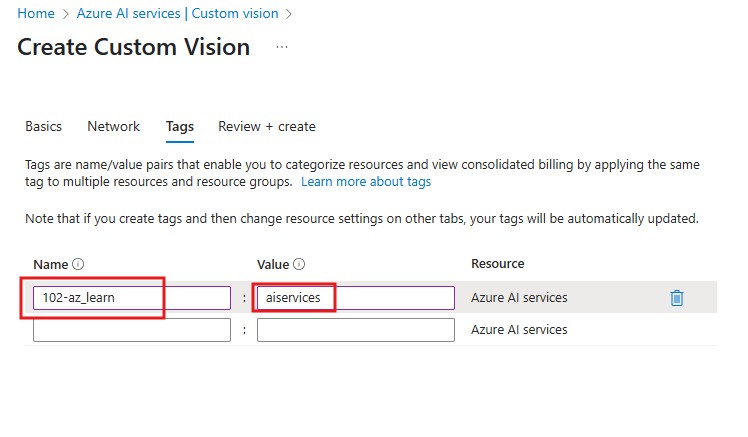
1. Now we will define our exceptions (exceptions.py) file:
2. Now that our exceptions are ready to be handled, we will have to go to our azure account and setup our custom vision training and prediction resources separately.
3. Click on Custom Vision
4.  Click on [Create] or [Create Custom Vision]:

Now we Fill In the basic informations for the training custom vision model, Why because I thought it be better if we have both resources differently. So here we first created our training resources and you can create the prediction resources the same way .

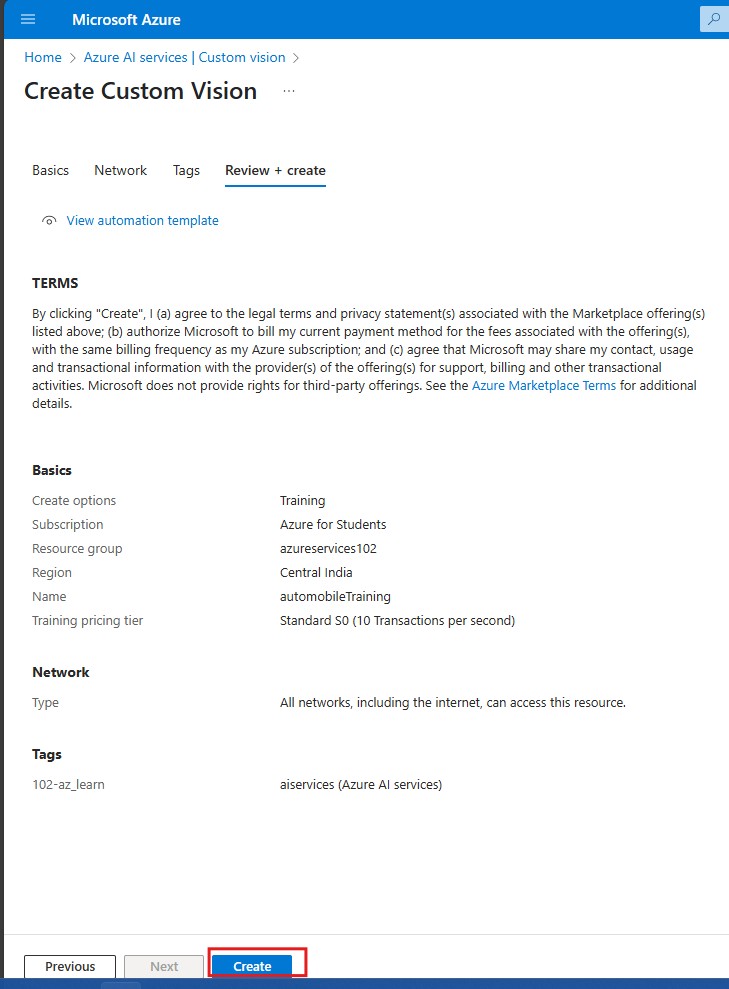


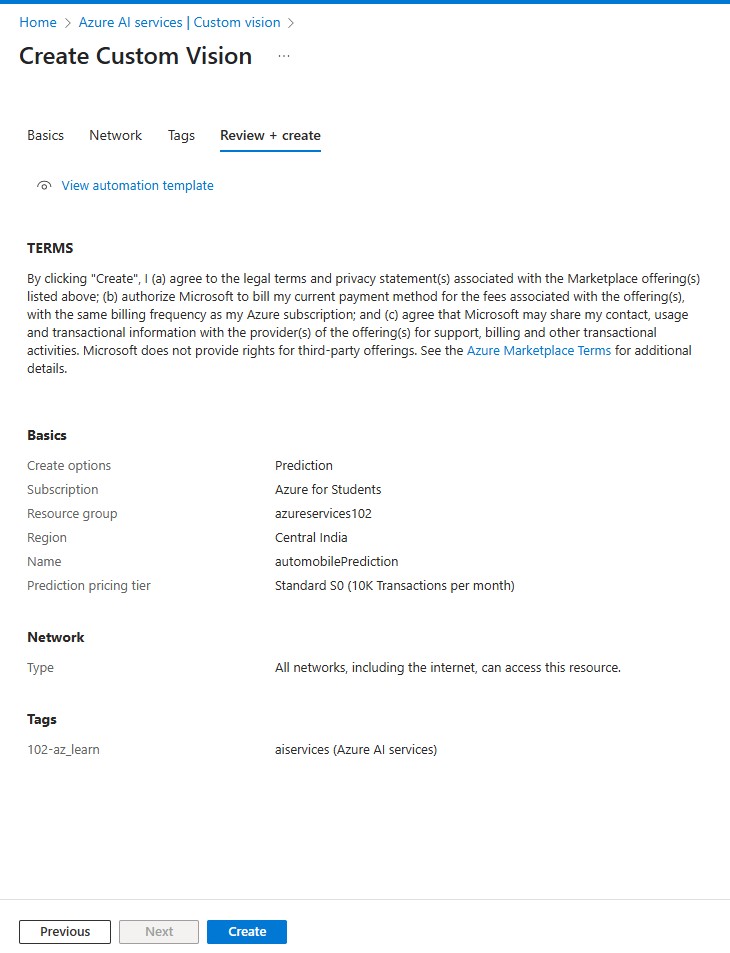
After you click on Next You will have to opt out your network sharing, I choose all so anyone can access and view it as well :



Now’s the part where we add tags to the resources, I have already made custom tags so I am using that here,:

Now Click on [ next + Preview ] and click on [create]:

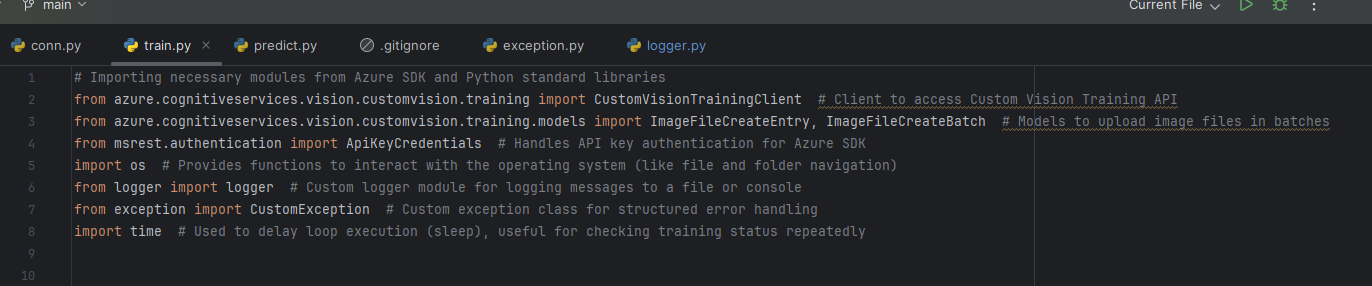


After this you **repeat** the **same** process **for** the **prediction** resources**.**

Train.py

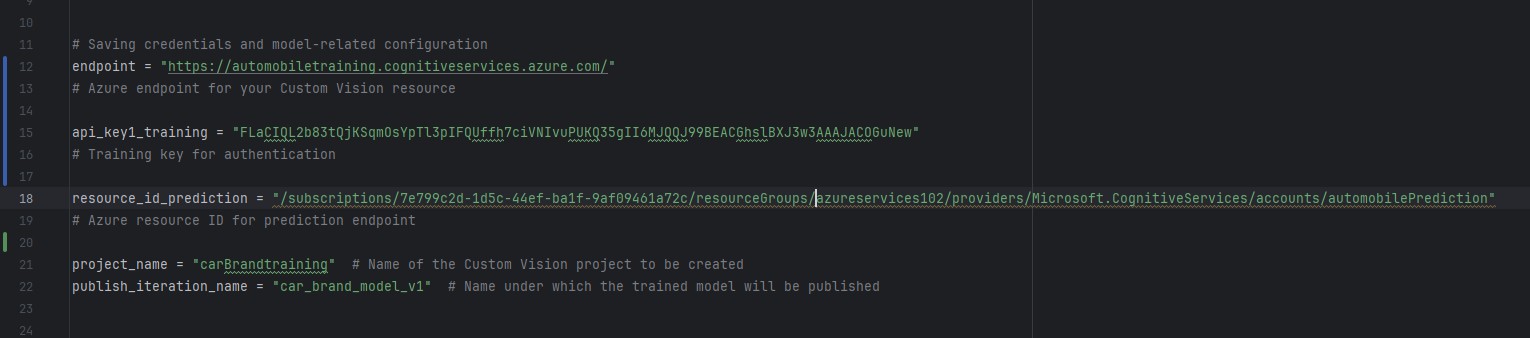
In the train.py python file what we will be doing is we will upload the images from the folder into the custom vision and train the model in this python code with sdk api of cognitive services of custom vision.

1. Part I:

First we will import the necessaries libraries from python

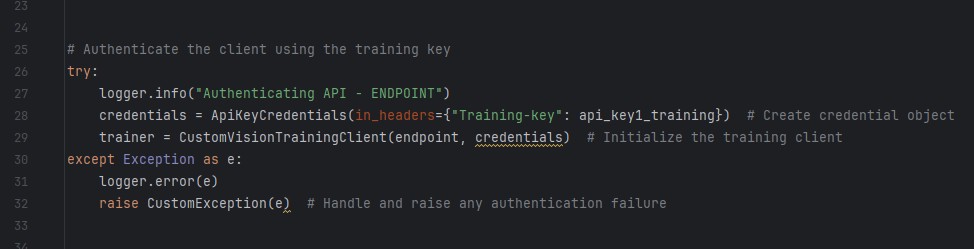
1. Part II

I hope that you saved your credentials like me

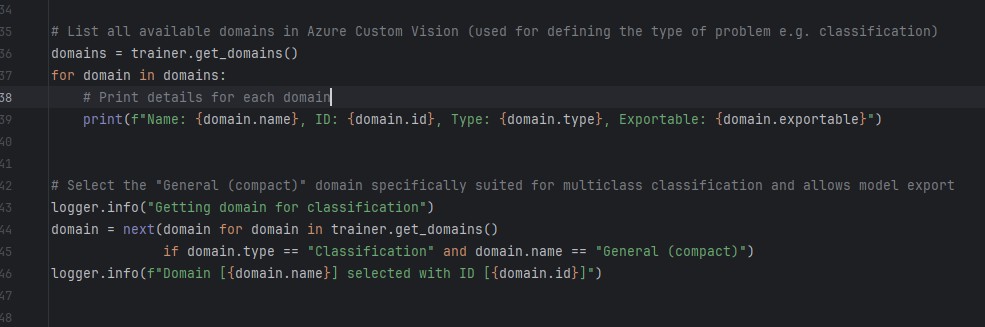
Endpoints | ApiKey\_Training | Resource ID for Prediction

1. Part III :

Use Microsoft msrest library and object of ApiKeyCredentials() method to pass in the valid id’s for the resources .



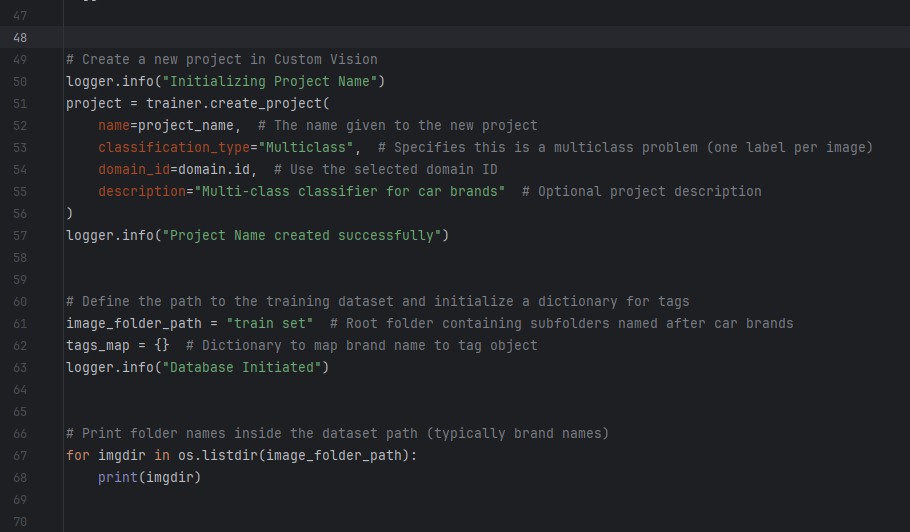
1. Part IV

Here we choose the domain that we want our model to be for, because we need a single for each single individual image so we will probably have to move forwards with Classification type and that too general single tag classification.

1. Part V

Since we have our folders separated as folder classified training it will be easier for us to tag the images in group using for loops.

Therefore we will now define our project with project name which we have already inside a string named project\_name, and assign the folder path of training data to another variable called image\_folder\_path.



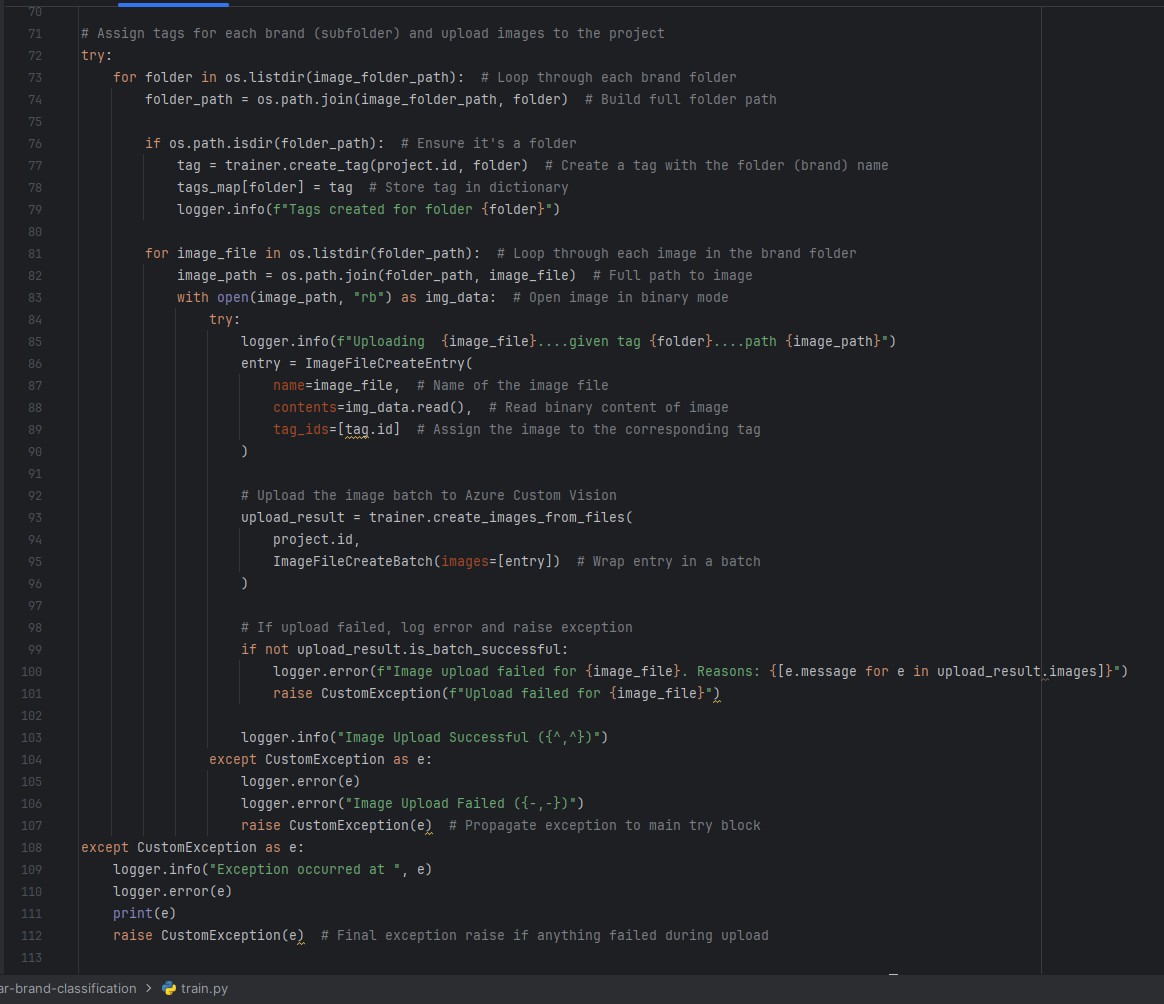
1. Part VI : IMP

In this part which is the most important part here we insert logger info path to every line to ensure in which part does the code portrays any error and in the log in file we will able to view them, as mentioned in the exception handling file date time line number error details and more.

So what this code does is , in short “It scans the folder in which we specified the folder path to a variable to training data location, so it will scan each image by their folder and tag each image by the folder name in which they are in ”

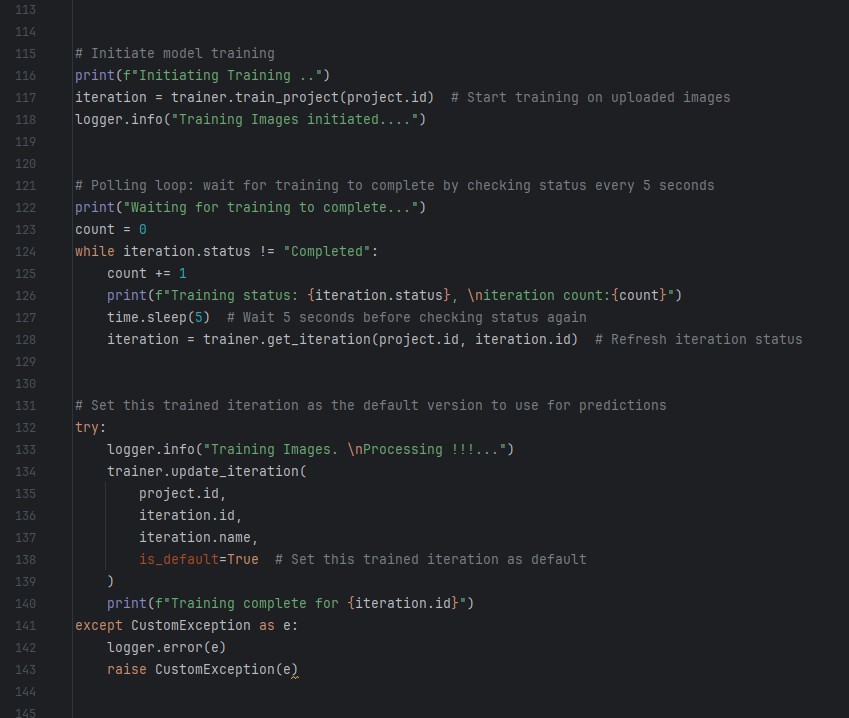
And read the image as byte format and upload them with tag into custom vision.

So the logger information is passed through each line of code if incase any part of the code raises an exception.



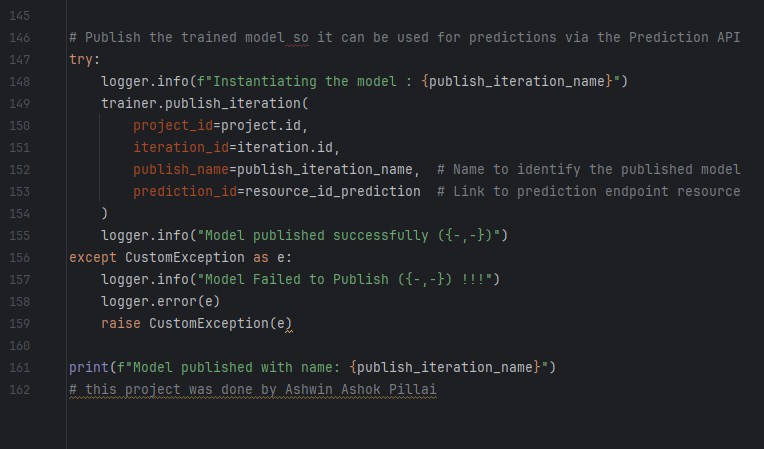
1. Part VII

Now we will train the data with the model that we have chosen earlier and that is multi classification with single tag per image model will be initialized for training, we can know when the training process is finished through a while loop and iteratively checks if training is completed or not, if not it will display that it has not in saying training number and with a hold of 5 seconds it checks.

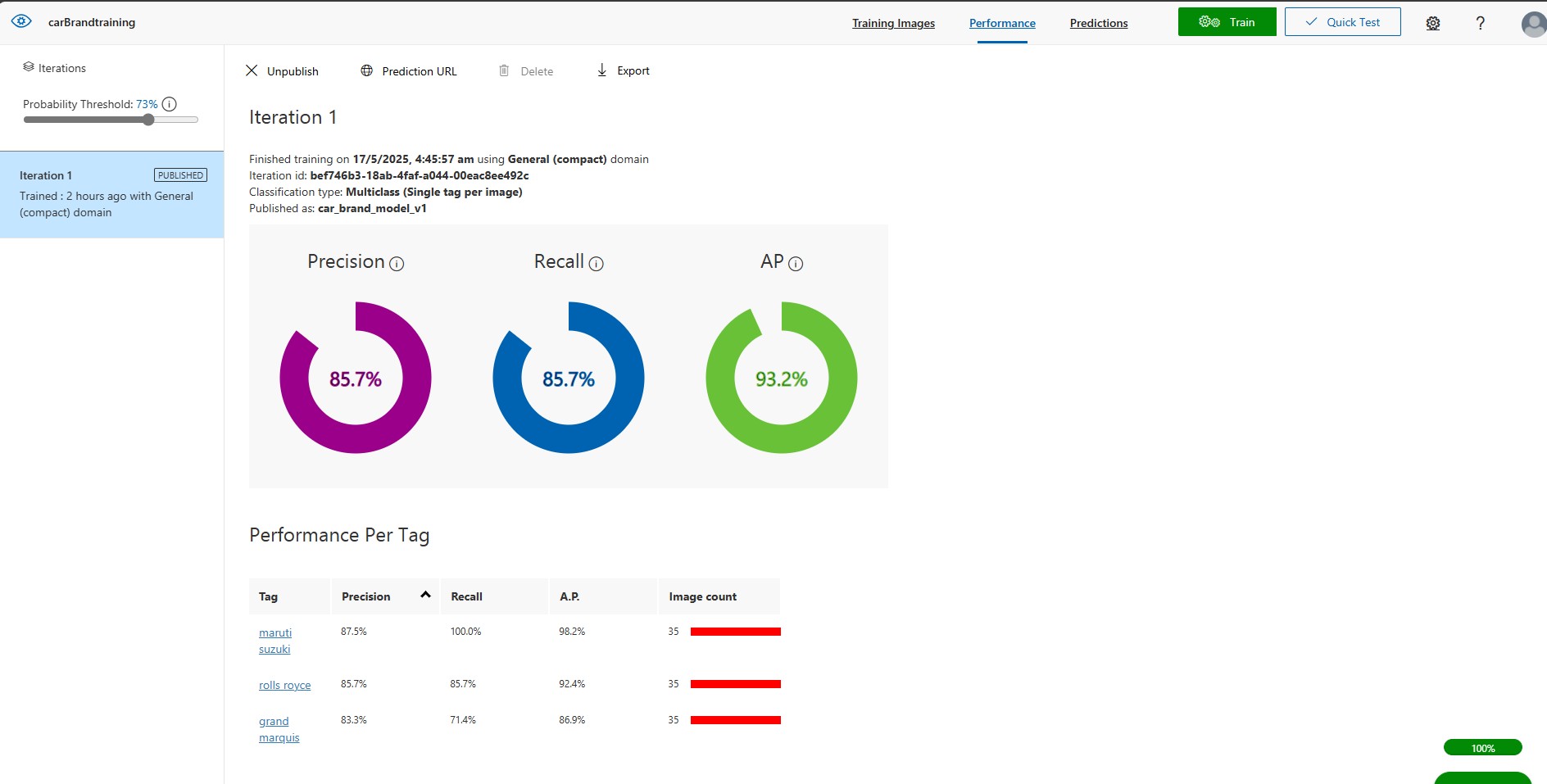


1. Part VIII

Now we have our model completed the training now and now we will move onto publish the model onto the custom vision.

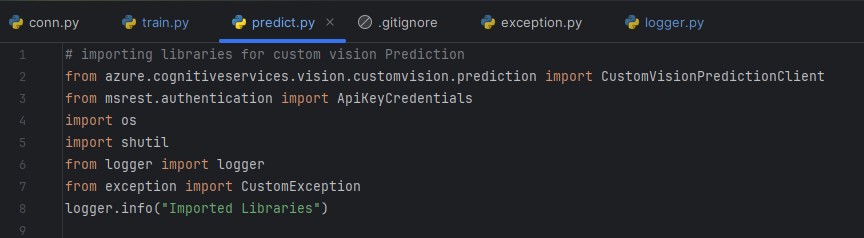
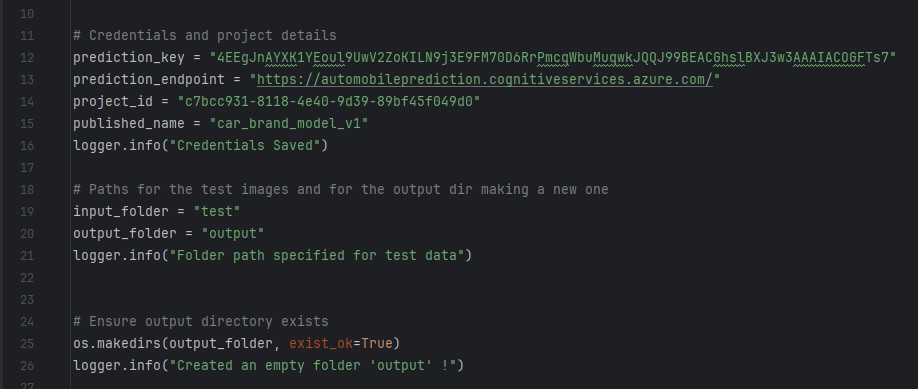
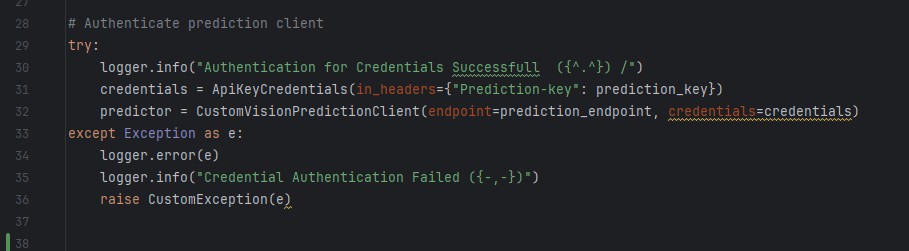
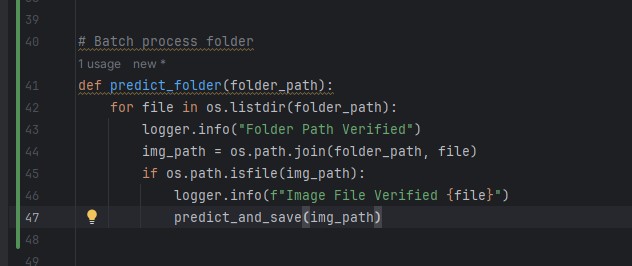
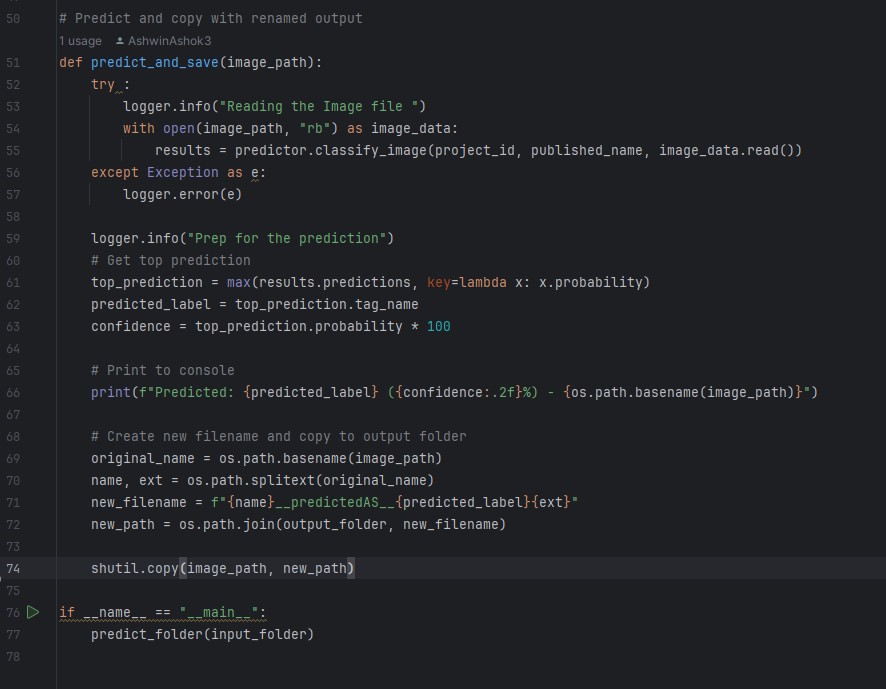


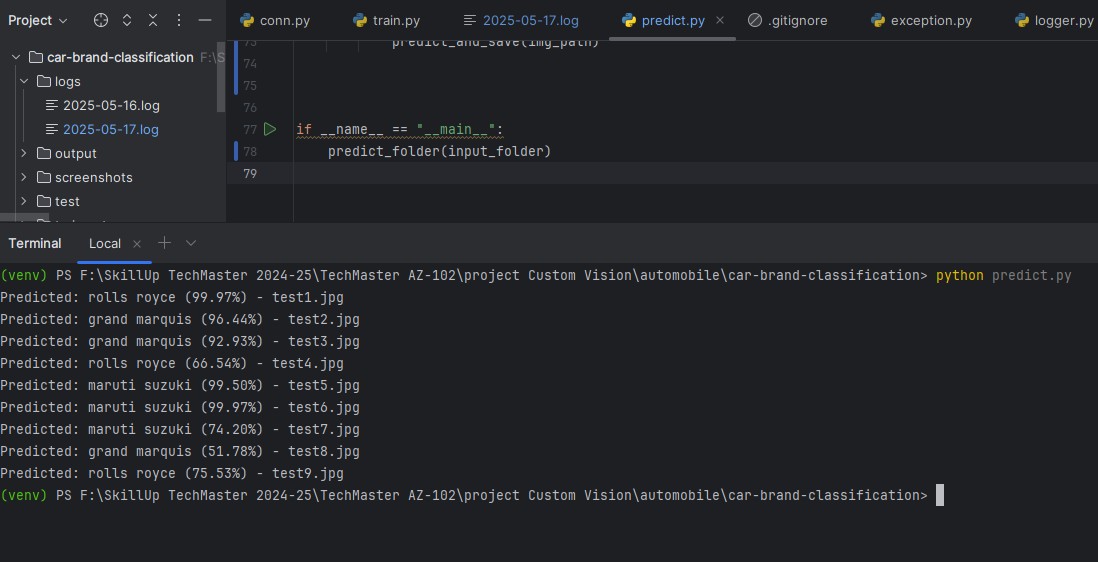
Now we have our model ready for prediction but before that let us see the performance of the model as well and evaluate the metrics of the model from the custom vision web app.



Predict.py

In this python file we will deal with prediction related resources same as before we import the same libraries like OS, azure cognitiveservies, logger file, exception file. But here there is a twist, we will now output our test images with the appropriate tags to their names to understand in which car brand has the model finally branded the car as and if incase it has landed a wrong tag to the wrong image we will be evidently aware of it as we know which car brand that particular car belonged to. Here we also define 2 functions which will predict the folder for the image and another folder for predict and saving the image.

1. Importing Libraries:
2. Credentials Initializing
   1. Paths for the output folder and predict data folder as well,
   2. Ensures if the output directory exists
3. Now we will authenticate the prediction client with the custom exception error handling
4. Function definition for the batch identification of the folders
5. Function for prediction of the image and saving it according to the tags that are provided in the code.

FINAL OUTPUT

Let us Also see it in custom vision how are the prediction and tags classification there, mostly it is same only but just to cross verify we will go through it again.

