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.

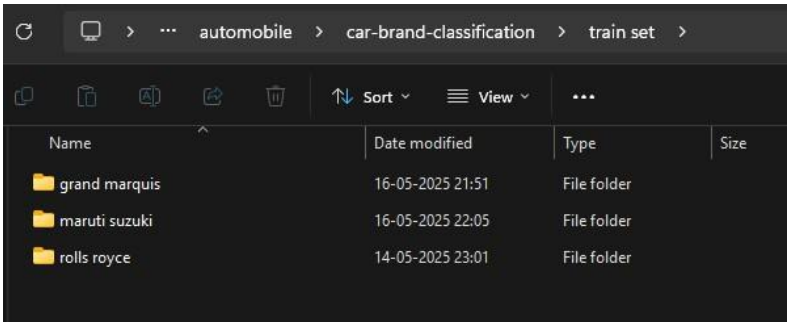
PERSONAL SOLUTION

I personally have chosen 3 different car brands

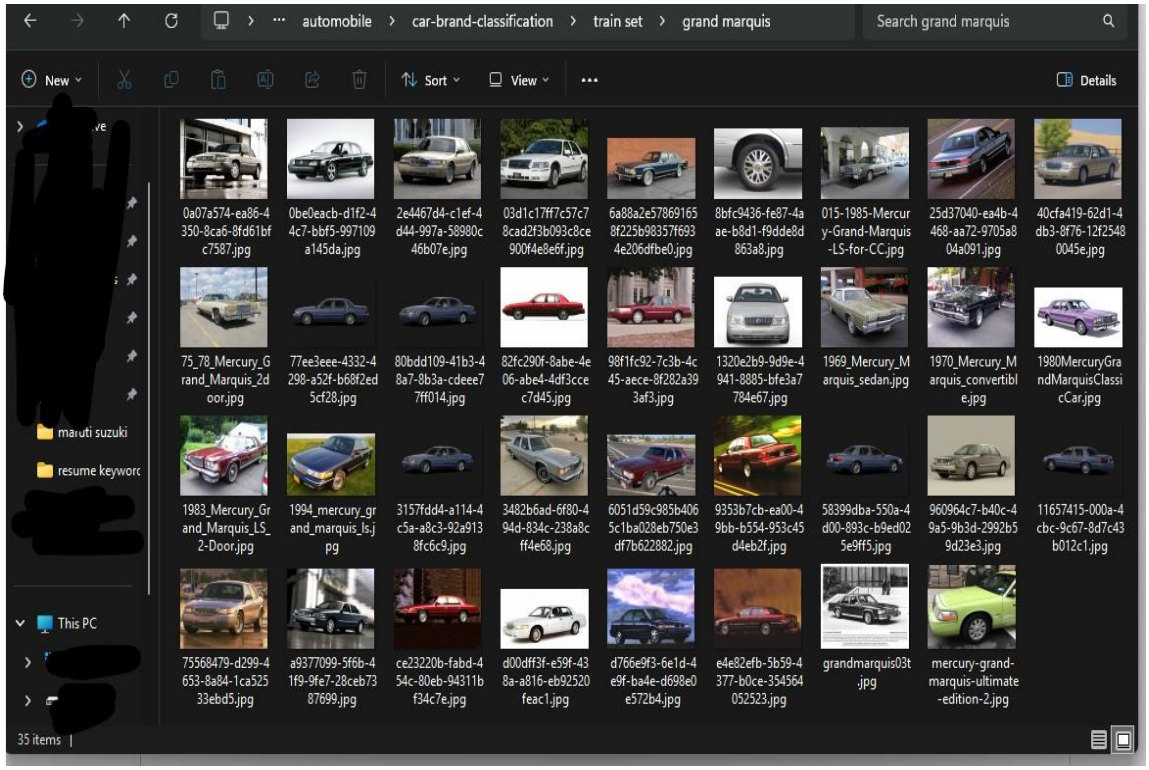
1. Mercury Grand Marquis
2. Maruti Suzuki
3. Rolls Royce

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.



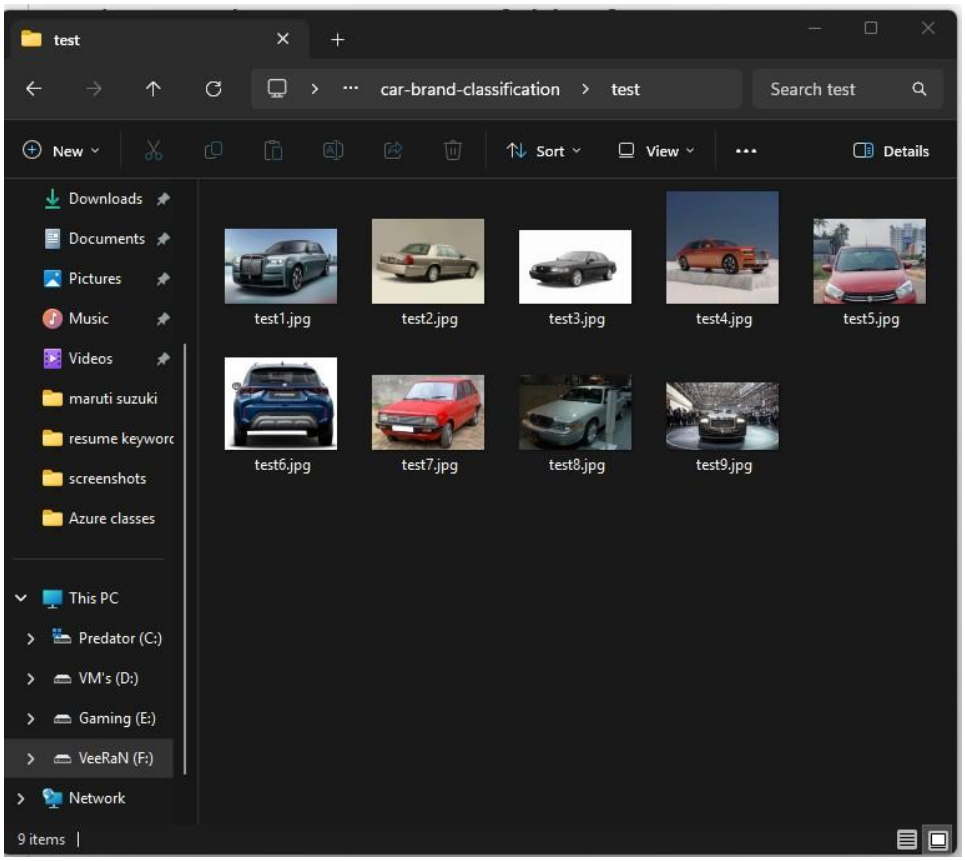
Since each of these folders contain 35 images, each.



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

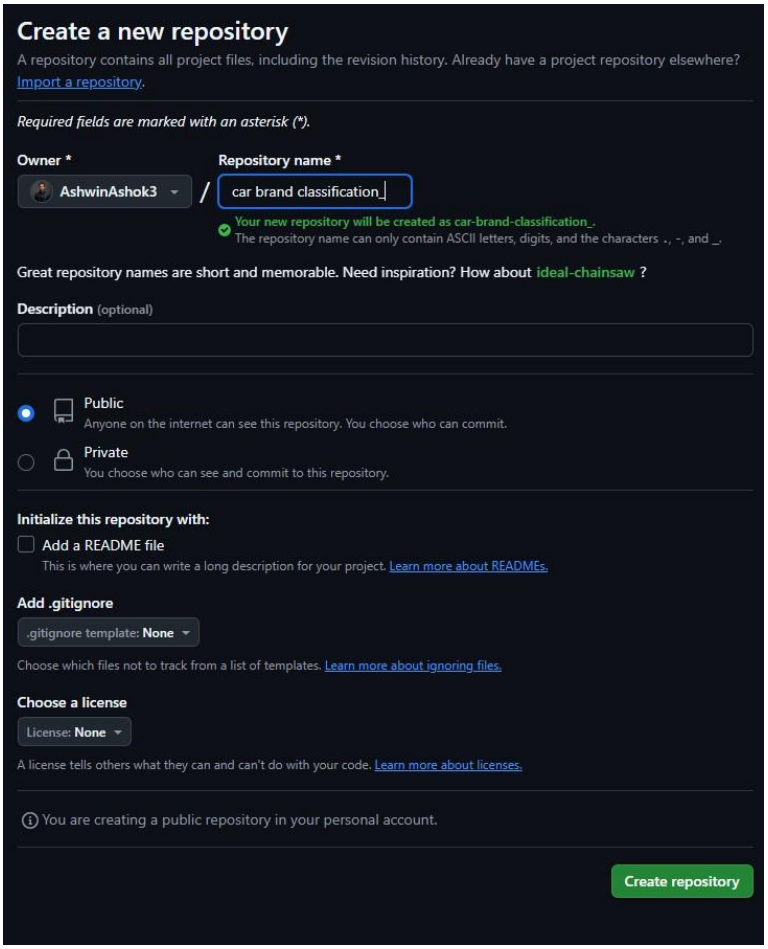
should also know the same so for testing and prediction we will have to keep a separate folder for our test set.



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.

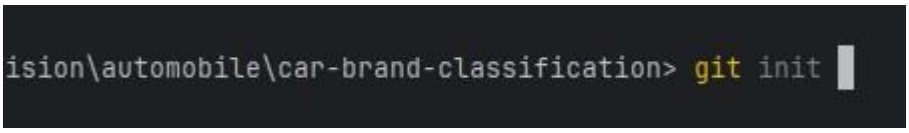
Also we will have to add “.gitignore” file other while running azure SDK’s it will show named too long can’t push or some error like that.

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>"

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.

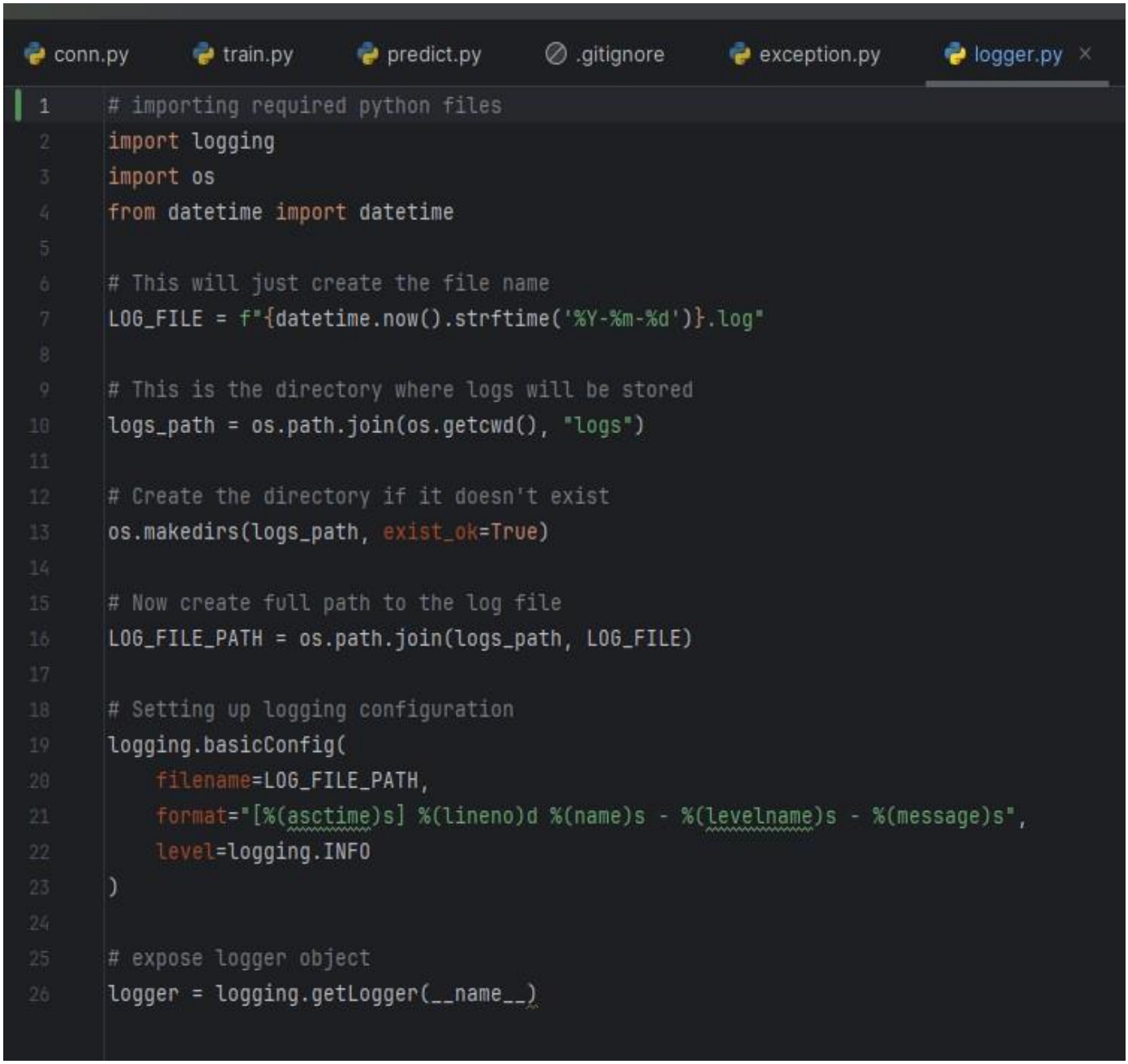
a. git add .
b. git status
c. git commit -m "description for the commit to happen"
d. git push -u origin main

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



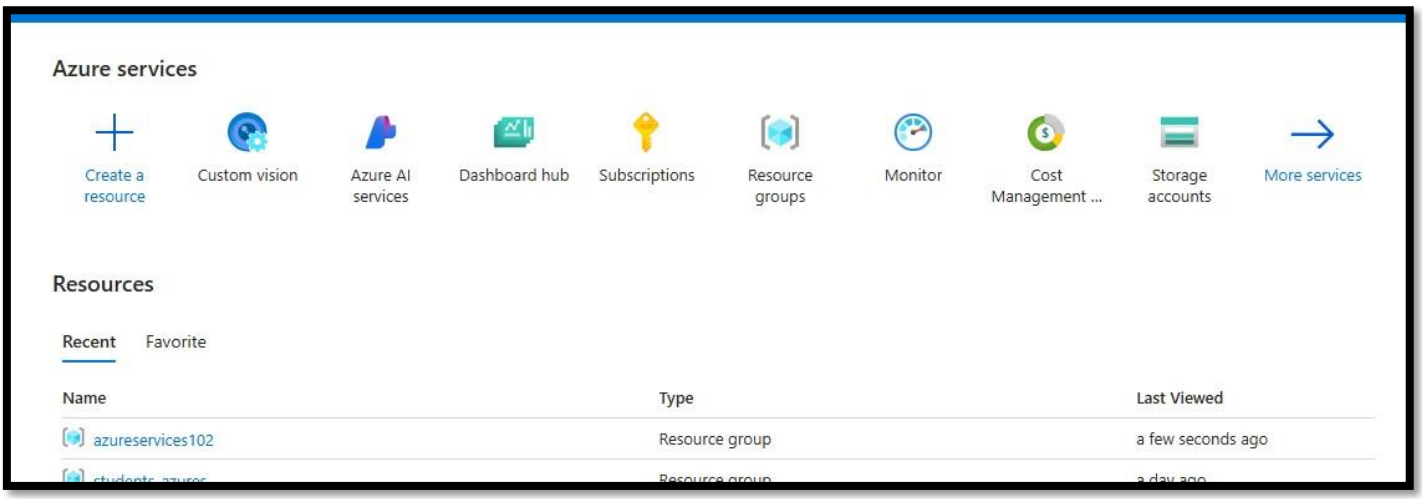
6. Now we will define our exceptions (exceptions.py) file:

conn.py | train.py | predict.py | .gitignore | exception.py | logger.py

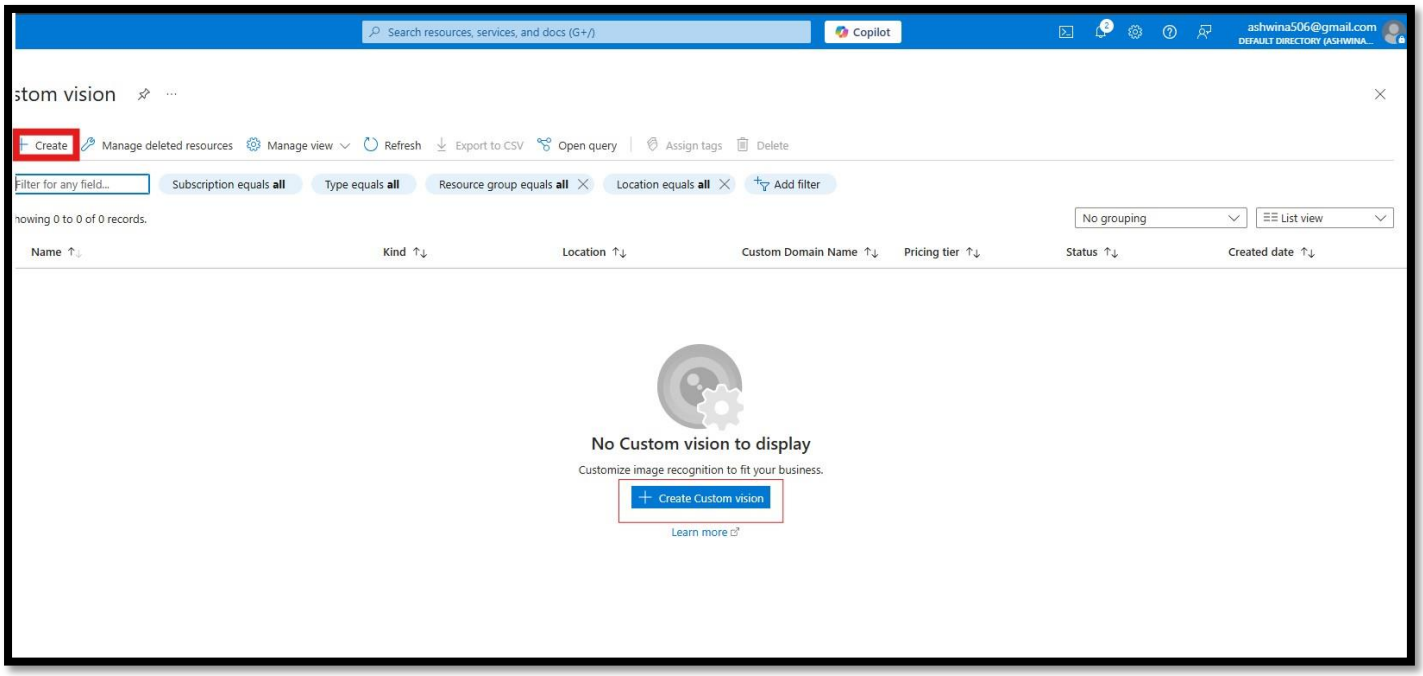
```python
# Purpose: Custom Exception Handling with detailed error tracking.

import sys  # To fetch runtime exception details like traceback
from logger import logger  #  Import logger.py
# Error message formatting function
1 usage  AshwinAshok3
def error_message_detail(error, error_detail: sys):
    _, _, exc_tb = error_detail.exc_info()  # Extract traceback info
    file_name = exc_tb.tb_frame.f_code.co_filename  # Which file caused the error
    line_number = exc_tb.tb_lineno  # Line number of error

    # Formatted error message
    error_message = (
        f"Error occurred in Python script: [{file_name}], "
        f"Line Number: [{line_number}], "
        f"Error Details: [{str(error)}]"
    )
    #  Log the error
    logger.error(error_message)

    return error_message


# Custom Exception Class
13 usages  AshwinAshok3
class CustomException(Exception):
    AshwinAshok3
    def __init__(self, error_message, error_detail: sys):
        super().__init__(error_message)  # Initialize base Exception class
        self.error_message = error_message_detail(error_message, error_detail)

    AshwinAshok3
    def __str__(self):
        return self.error_message  # When printed

    AshwinAshok3
    def __repr__(self):
        return self.error_message  # For object representation
```

7. 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.

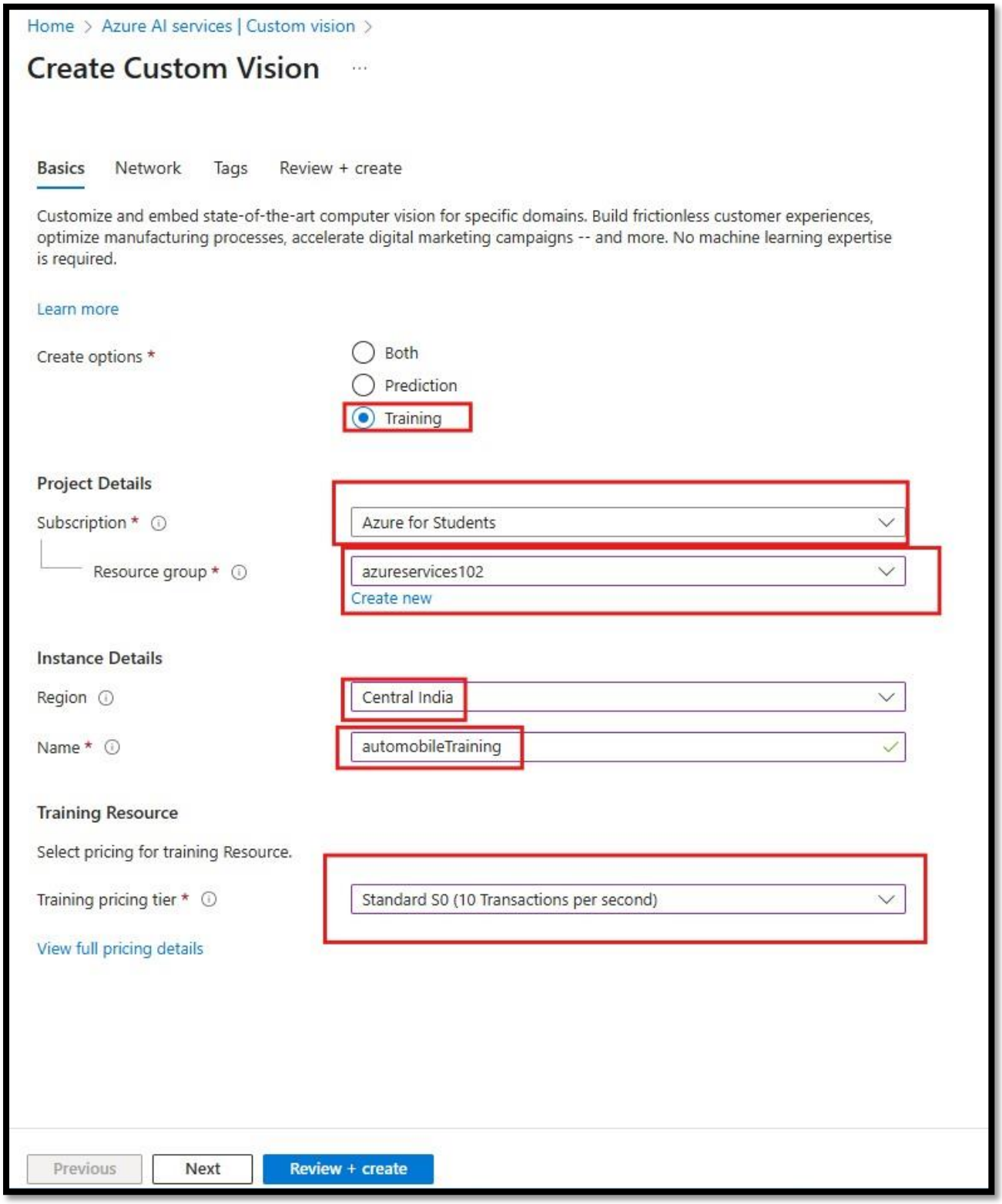
8. Click on Custom Vision



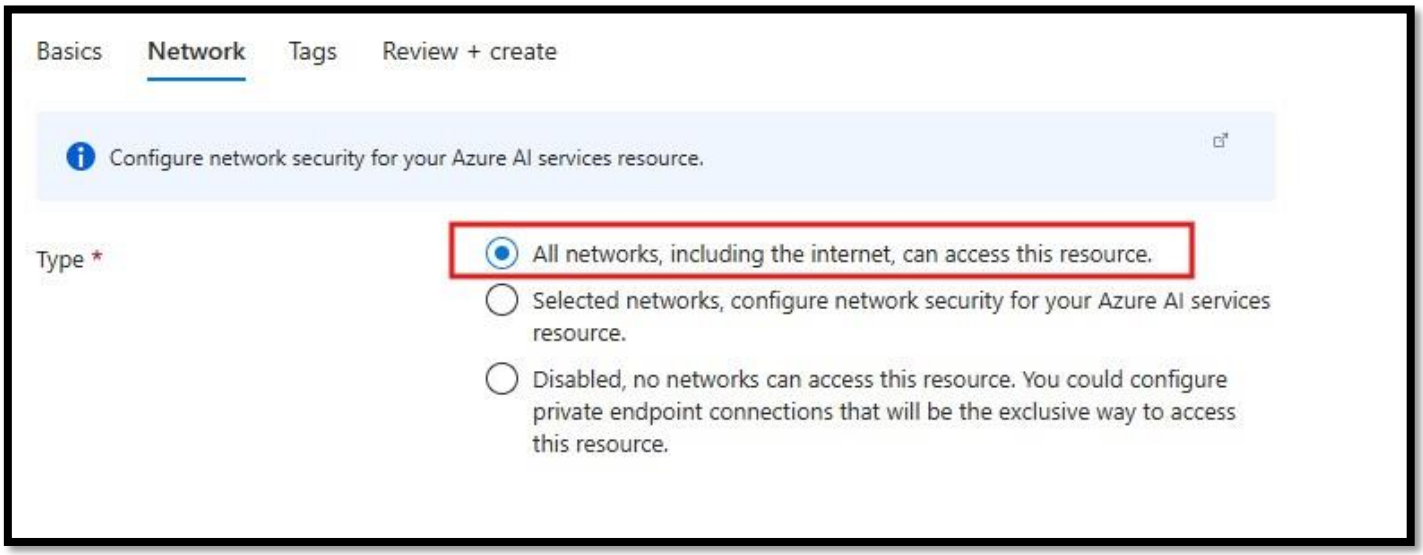
9. Click on [Create] or [Create Custom Vision]:



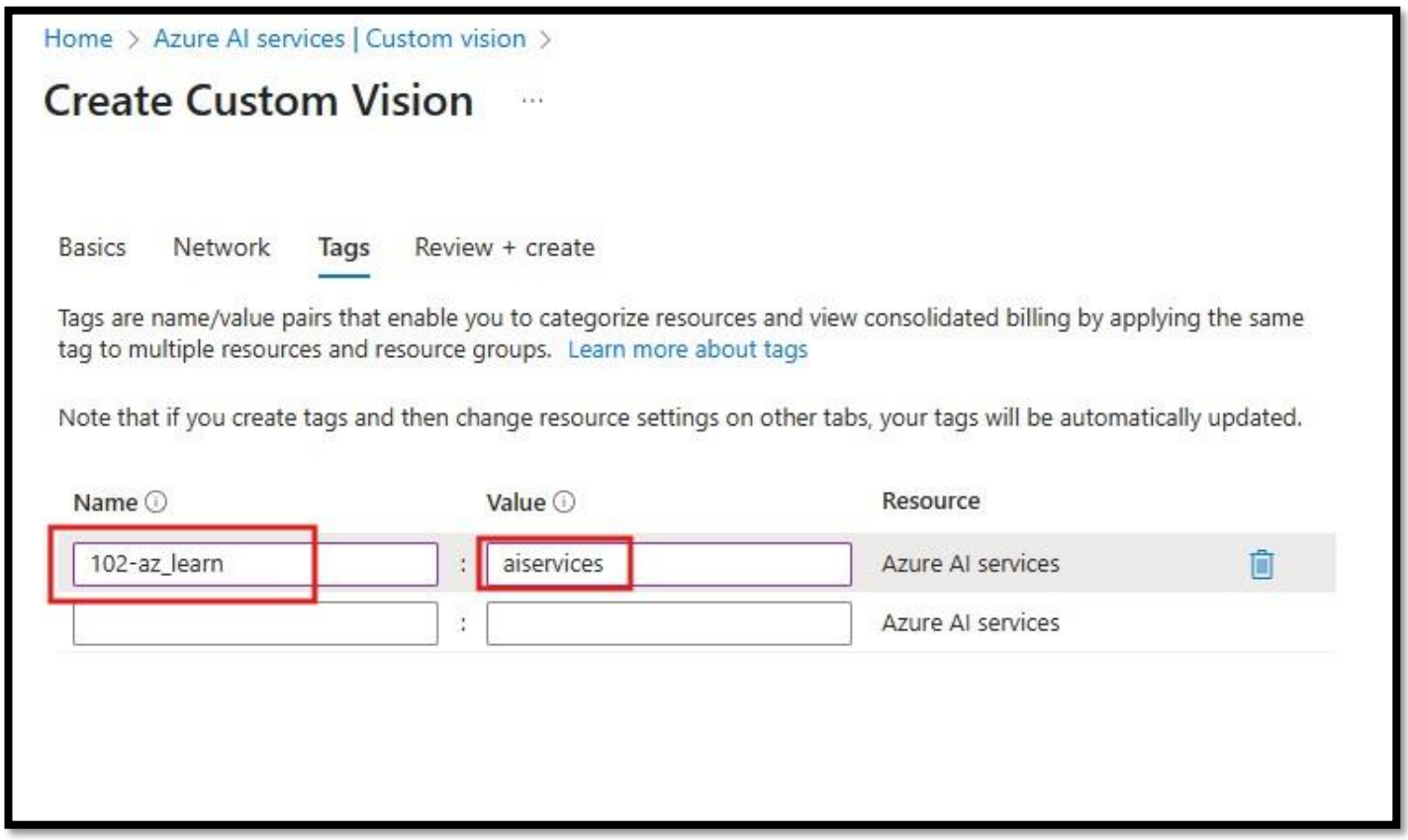
Now we Fill In the basic informations for the training custom vision model, Why because I thought it be better if we have both resources differently. So here we first created our training resources and you can create the prediction resources the same way .



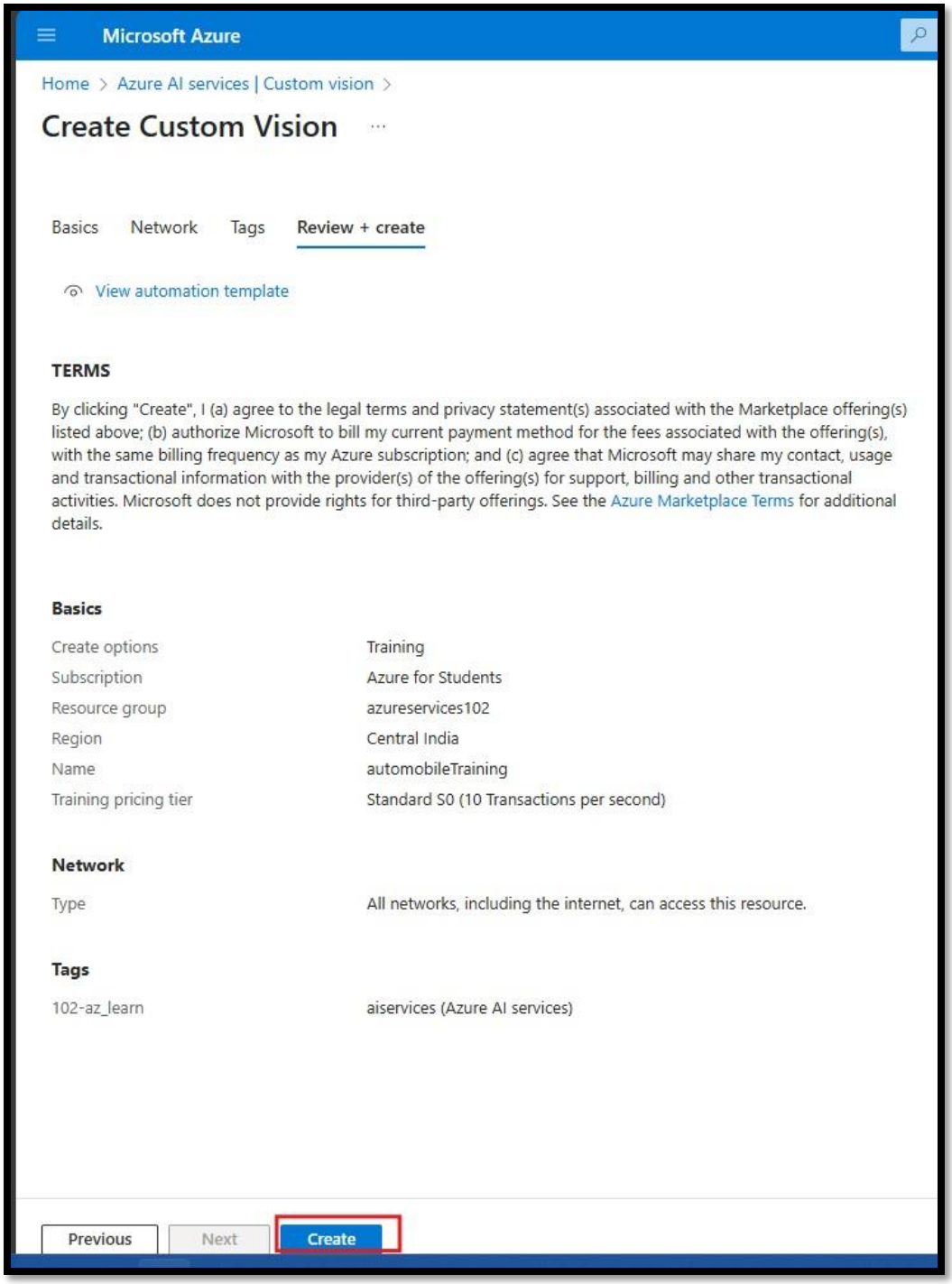
After you click on Next You will have to opt out your network sharing, I choose all so anyone can access and view it as well :



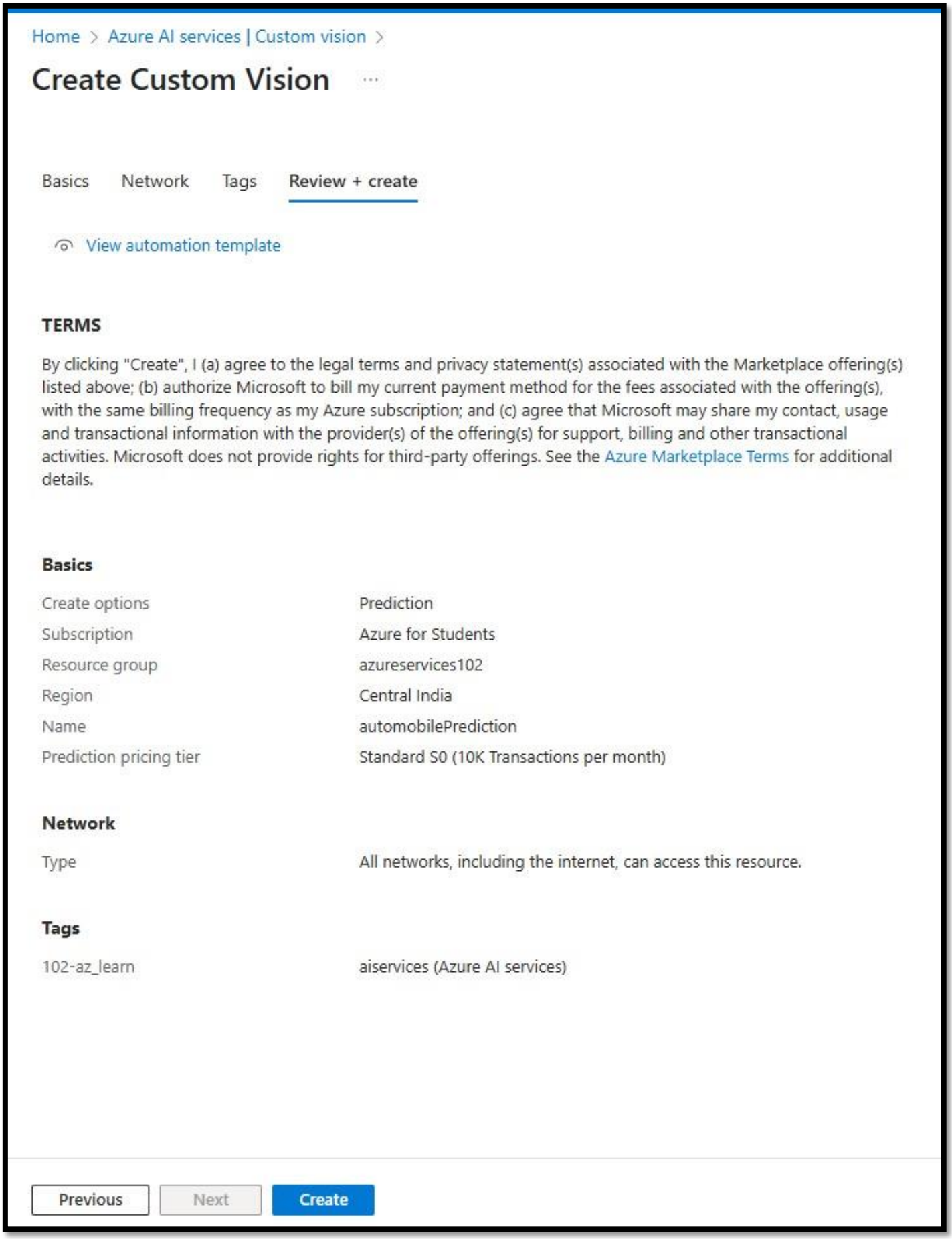
Now's the part where we add tags to the resources, I have already made custom tags so I am using that here,:



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



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

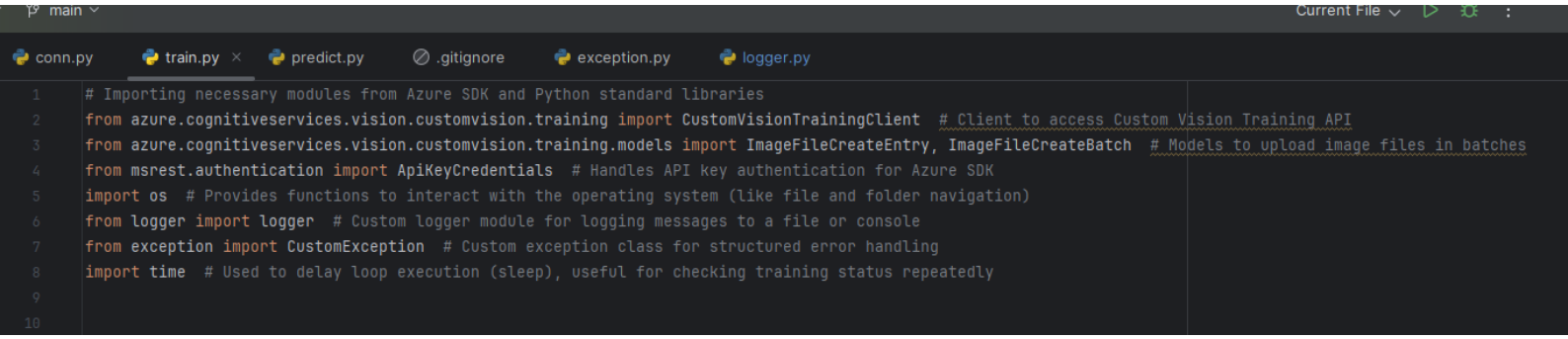


TRAIN.PY

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



2. Part II

I hope that you saved your credentials like me

Endpoints | ApiKey_Training | Resource ID for Prediction

3. Part III :

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

```
23
24
25      # Authenticate the client using the training key
26      try:
27          logger.info("Authenticating API - ENDPOINT")
28          credentials = ApiKeyCredentials(in_headers={"Training-key": api_key1_training})  # Create credential object
29          trainer = CustomVisionTrainingClient(endpoint, credentials)  # Initialize the training client
30      except Exception as e:
31          logger.error(e)
32          raise CustomException(e)  # Handle and raise any authentication failure
33
```

4. 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.

```
34
35      # List all available domains in Azure Custom Vision (used for defining the type of problem e.g. classification)
36      domains = trainer.get_domains()
37      for domain in domains:
38          # Print details for each domain
39          print(f"Name: {domain.name}, ID: {domain.id}, Type: {domain.type}, Exportable: {domain.exportable}")
40
41
42      # Select the "General (compact)" domain specifically suited for multiclass classification and allows model export
43      logger.info("Getting domain for classification")
44      domain = next(domain for domain in trainer.get_domains()
45                    if domain.type == "Classification" and domain.name == "General (compact)")
46      logger.info(f"Domain [{domain.name}] selected with ID [{domain.id}]")
47
48
```

5. 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.

```
47
48
49  # Create a new project in Custom Vision
50  logger.info("Initializing Project Name")
51  project = trainer.create_project(
52      name=project_name,  # The name given to the new project
53      classification_type="Multiclass",  # Specifies this is a multiclass problem (one label per image)
54      domain_id=domain.id,  # Use the selected domain ID
55      description="Multi-class classifier for car brands"  # Optional project description
56  )
57  logger.info("Project Name created successfully")
58
59
60  # Define the path to the training dataset and initialize a dictionary for tags
61  image_folder_path = "train set"  # Root folder containing subfolders named after car brands
62  tags_map = {}  # Dictionary to map brand name to tag object
63  logger.info("Database Initiated")
64
65
66  # Print folder names inside the dataset path (typically brand names)
67  for imgdir in os.listdir(image_folder_path):
68      print(imgdir)
69
70
```

6. 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.

```python
# Assign tags for each brand (subfolder) and upload images to the project
try:
    for folder in os.listdir(image_folder_path):  # Loop through each brand folder
        folder_path = os.path.join(image_folder_path, folder)  # Build full folder path

        if os.path.isdir(folder_path):  # Ensure it's a folder
            tag = trainer.create_tag(project.id, folder)  # Create a tag with the folder (brand) name
            tags_map[folder] = tag  # Store tag in dictionary
            logger.info(f"Tags created for folder {folder}")

        for image_file in os.listdir(folder_path):  # Loop through each image in the brand folder
            image_path = os.path.join(folder_path, image_file)  # Full path to image
            with open(image_path, "rb") as img_data:  # Open image in binary mode
                try:
                    logger.info(f"Uploading  {image_file}....given tag {folder}....path {image_path}")
                    entry = ImageFileCreateEntry(
                        name=image_file,  # Name of the image file
                        contents=img_data.read(),  # Read binary content of image
                        tag_ids=[tag.id]  # Assign the image to the corresponding tag
                    )

                    # Upload the image batch to Azure Custom Vision
                    upload_result = trainer.create_images_from_files(
                        project.id,
                        ImageFileCreateBatch(images=[entry])  # Wrap entry in a batch
                    )

                    # If upload failed, log error and raise exception
                    if not upload_result.is_batch_successful:
                        logger.error(f"Image upload failed for {image_file}. Reasons: {[e.message for e in upload_result.images]}")
                        raise CustomException(f"Upload failed for {image_file}")

                    logger.info("Image Upload Successful ({^,^})")
                except CustomException as e:
                    logger.error(e)
                    logger.error("Image Upload Failed ({-,-})")
                    raise CustomException(e)  # Propagate exception to main try block
except CustomException as e:
    logger.info("Exception occurred at ", e)
    logger.error(e)
    print(e)
    raise CustomException(e)  # Final exception raise if anything failed during upload

```

ar-brand-classification > train.py

7. 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.

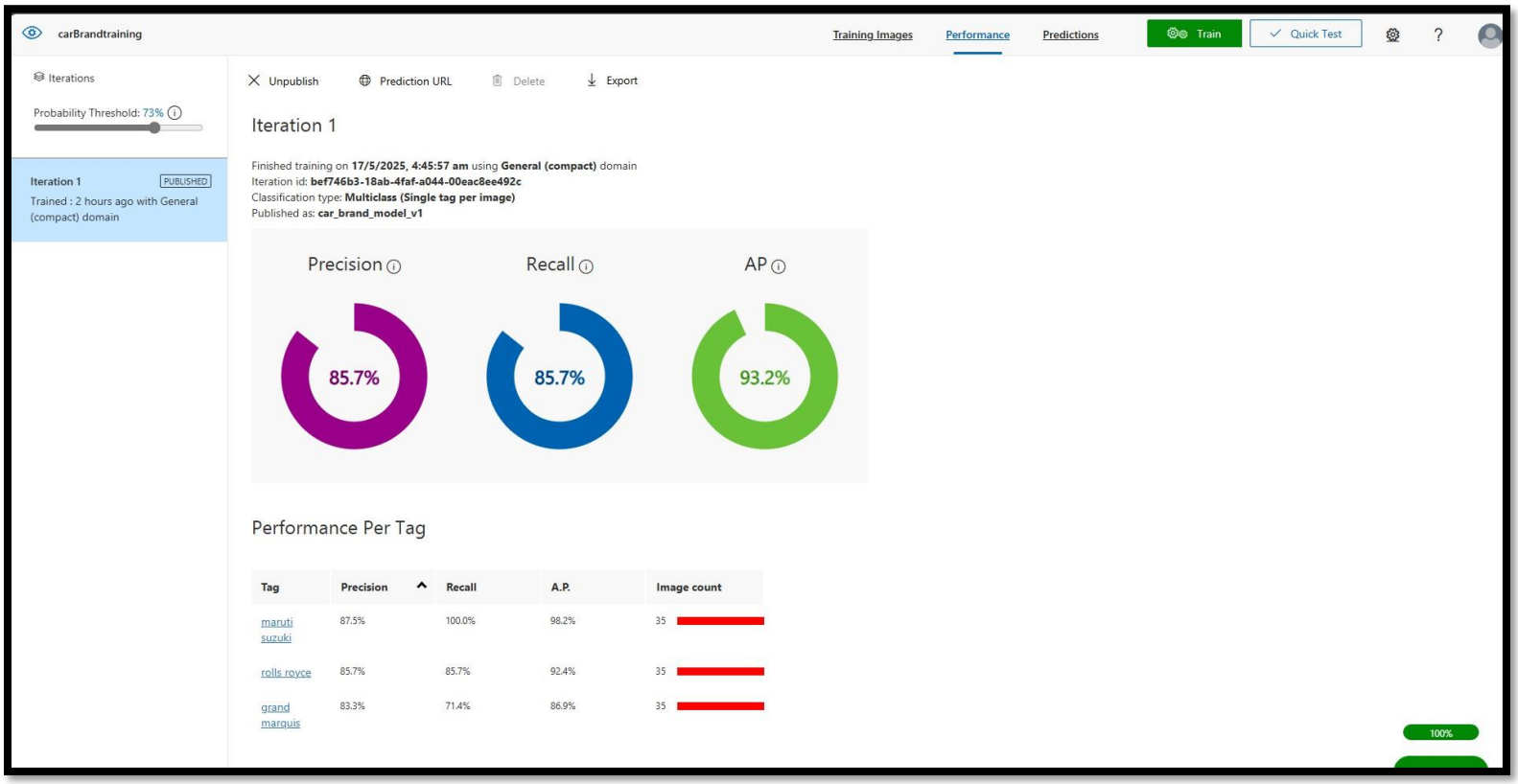
```python

# Initiate model training
print(f"Initiating Training ..")
iteration = trainer.train_project(project.id)  # Start training on uploaded images
logger.info("Training Images initiated....")


# Polling loop: wait for training to complete by checking status every 5 seconds
print("Waiting for training to complete...")
count = 0
while iteration.status != "Completed":
    count += 1
    print(f"Training status: {iteration.status}, \niteration count:{count}")
    time.sleep(5)  # Wait 5 seconds before checking status again
    iteration = trainer.get_iteration(project.id, iteration.id)  # Refresh iteration status


# Set this trained iteration as the default version to use for predictions
try:
    logger.info("Training Images. \nProcessing !!!...")
    trainer.update_iteration(
        project.id,
        iteration.id,
        iteration.name,
        is_default=True  # Set this trained iteration as default
    )
    print(f"Training complete for {iteration.id}")
except CustomException as e:
    logger.error(e)
    raise CustomException(e)


```

8. Part VIII

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

```python

# Publish the trained model so it can be used for predictions via the Prediction API
try:
    logger.info(f"Instantiating the model : {publish_iteration_name}")
    trainer.publish_iteration(
        project_id=project.id,
        iteration_id=iteration.id,
        publish_name=publish_iteration_name,  # Name to identify the published model
        prediction_id=resource_id_prediction  # Link to prediction endpoint resource
    )
    logger.info("Model published successfully ({-,-})")
except CustomException as e:
    logger.info("Model Failed to Publish ({-,-}) !!!")
    logger.error(e)
    raise CustomException(e)

print(f"Model published with name: {publish_iteration_name}")
# this project was done by Ashwin Ashok Pillai
```

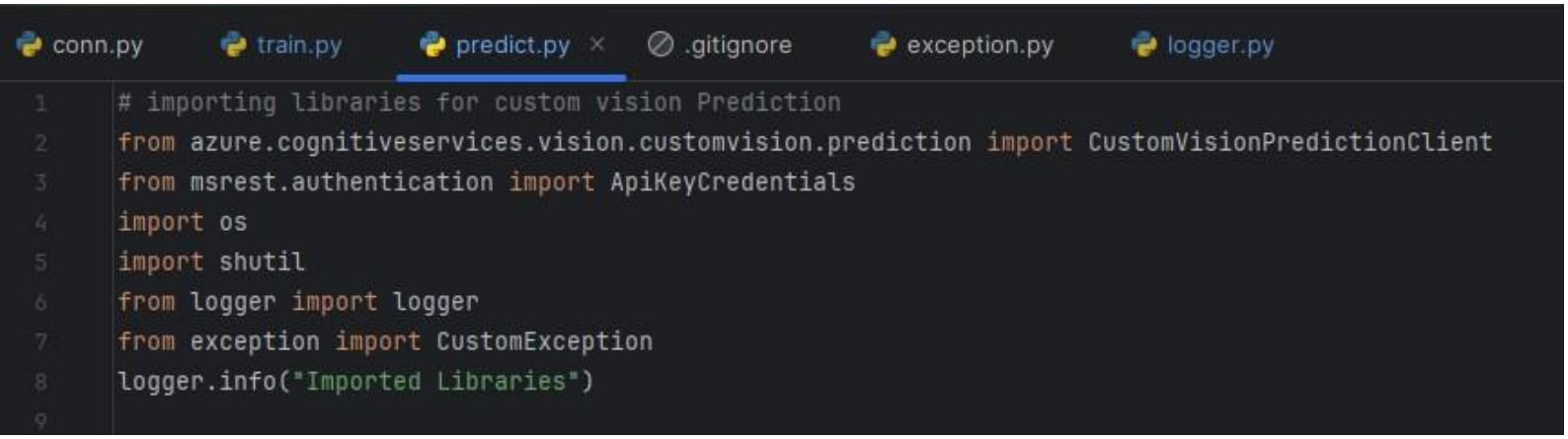
Now we have our model ready for prediction but before that let us see the performance of the model as well and evaluate the metrics of the model from the custom vision web app.



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.

1st. Importing Libraries:



2nd. Credentials Initializing

a. Paths for the output folder and predict data folder as well,

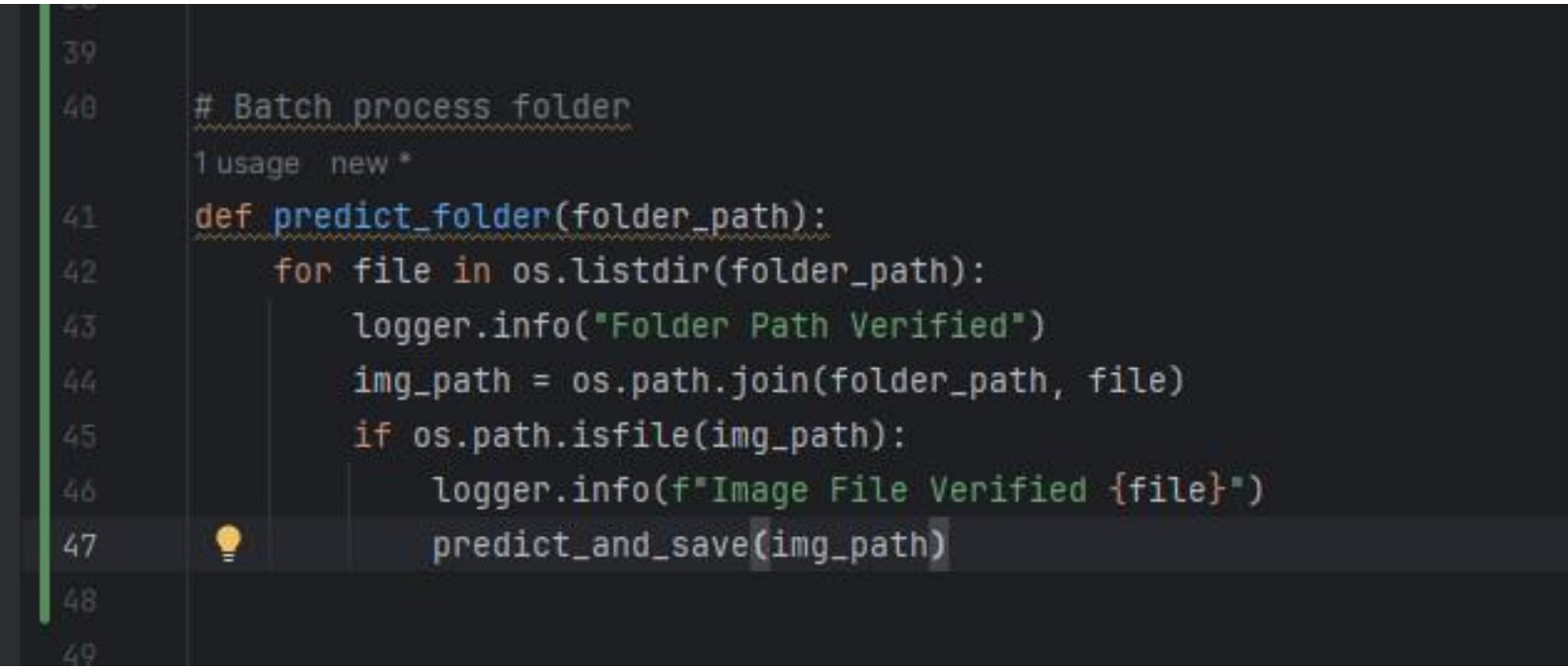
b. Ensures if the output directory exists

```
10
11  # Credentials and project details
12  prediction_key = "4EEgJnAYXK1YEoul9UwV2ZoKILN9j3E9FM70D6RrPmcqWbuMuqwkJQQJ99BEACGhslBXJ3w3AAAIACOGFTs7"
13  prediction_endpoint = "https://automobileprediction.cognitiveservices.azure.com/"
14  project_id = "c7bcc931-8118-4e40-9d39-89bf45f049d0"
15  published_name = "car_brand_model_v1"
16  logger.info("Credentials Saved")
17
18  # Paths for the test images and for the output dir making a new one
19  input_folder = "test"
20  output_folder = "output"
21  logger.info("Folder path specified for test data")
22
23
24  # Ensure output directory exists
25  os.makedirs(output_folder, exist_ok=True)
26  logger.info("Created an empty folder 'output' !")
27
```

3rd. Now we will authenticate the prediction client with the custom exception error handling

```
27
28  # Authenticate prediction client
29  try:
30      logger.info("Authentication for Credentials Successfull  ({^.^}) /")
31      credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
32      predictor = CustomVisionPredictionClient(endpoint=prediction_endpoint, credentials=credentials)
33  except Exception as e:
34      logger.error(e)
35      logger.info("Credential Authentication Failed ({-,-})")
36      raise CustomException(e)
37
38
```

4th. Function definition for the batch identification of the folders

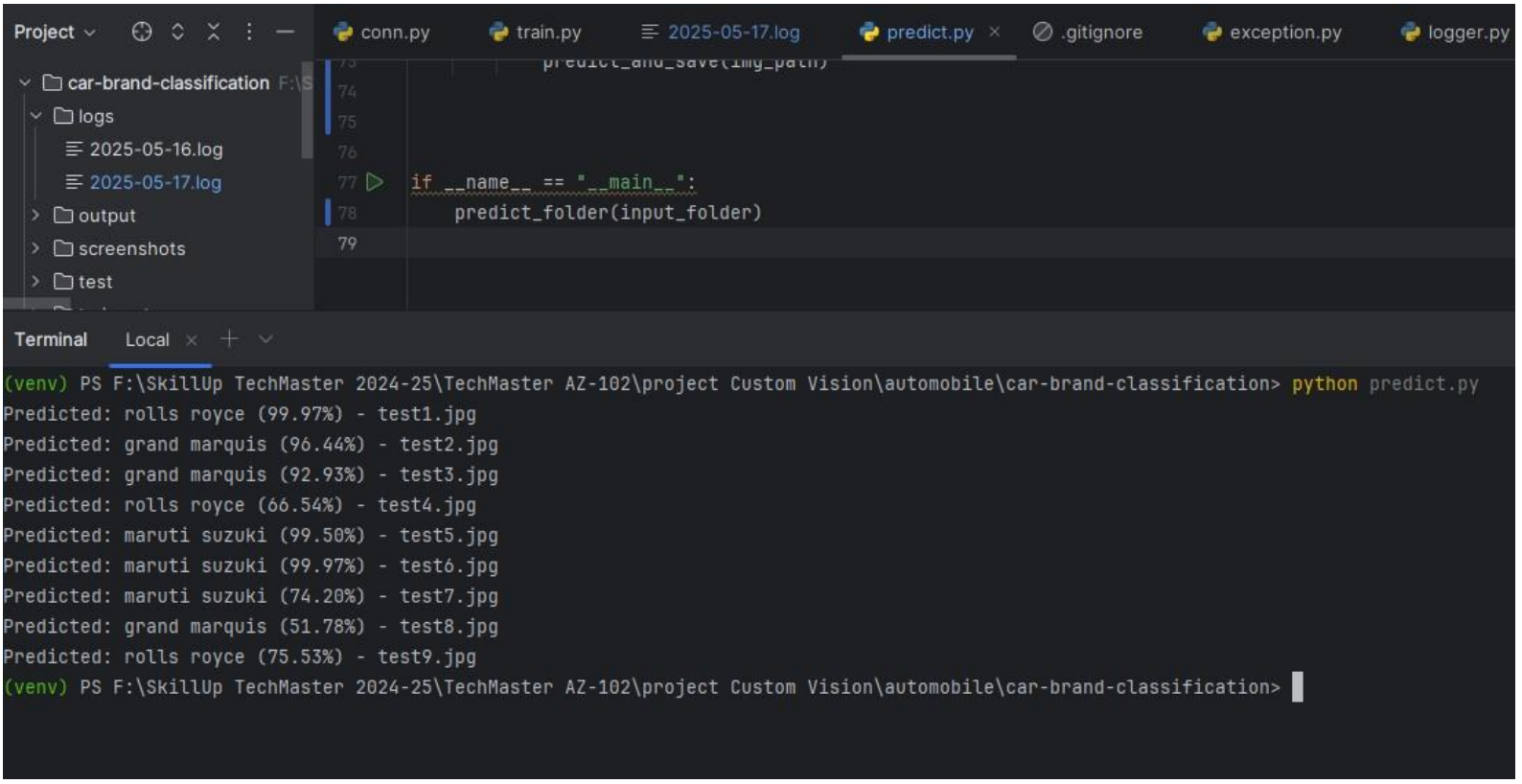


```
39
40  # Batch process folder
    1 usage  new *
41  def predict_folder(folder_path):
42      for file in os.listdir(folder_path):
43          logger.info("Folder Path Verified")
44          img_path = os.path.join(folder_path, file)
45          if os.path.isfile(img_path):
46              logger.info(f"Image File Verified {file}")
47              predict_and_save(img_path)
48
49
```

5th. Function for prediction of the image and saving it according to the tags that are provided in the code.

```python
# Predict and copy with renamed output
1 usage  AshwinAshok3
def predict_and_save(image_path):
    try :
        logger.info("Reading the Image file ")
        with open(image_path, "rb") as image_data:
            results = predictor.classify_image(project_id, published_name, image_data.read())
    except Exception as e:
        logger.error(e)

    logger.info("Prep for the prediction")
    # Get top prediction
    top_prediction = max(results.predictions, key=lambda x: x.probability)
    predicted_label = top_prediction.tag_name
    confidence = top_prediction.probability * 100

    # Print to console
    print(f"Predicted: {predicted_label} ({confidence:.2f}%) - {os.path.basename(image_path)}")

    # Create new filename and copy to output folder
    original_name = os.path.basename(image_path)
    name, ext = os.path.splitext(original_name)
    new_filename = f"{name}__predictedAS__{predicted_label}{ext}"
    new_path = os.path.join(output_folder, new_filename)

    shutil.copy(image_path, new_path)

if __name__ == "__main__":
    predict_folder(input_folder)
```

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.

49d0#/predictions

My Tags

Add a tag and press enter

Predictions

Tag	Probability
grand marquis	51.7%
maruti suzuki	45.5%
rolls royce	2.6%

Save and close

Image Detail

My Tags

Add a tag and press enter

Predictions

Tag	Probability
rolls royce	66.5%
grand marquis	32.9%
maruti suzuki	0.4%

Save and close

My Tags

Add a tag and press enter

Predictions

Tag	Probability
maruti suzuki	74.1%
grand marquis	22.6%
rolls royce	3.1%

Save and close

My Tags

Add a tag and press enter

Predictions

Tag	Probability
rolls royce	75.5%
grand marquis	24.3%
maruti suzuki	0%

Save and close

Predictions

Tag	Probability
grand marquis	92.9%
rolls royce	6.9%
maruti suzuki	0.1%

Save and close

My Tags

Add a tag and press enter

Predictions

Tag	Probability
grand marquis	96.4%
rolls royce	3.2%
maruti suzuki	0.3%

Save and close

predictions

My Tags

Add a tag and press enter

Predictions

Tag	Probability
maruti suzuki	99.5%
grand marquis	0.4%
rolls royce	0%

Save and close

My Tags

Add a tag and press enter

Predictions

Tag	Probability
rolls royce	99.9%
maruti suzuki	0%
grand marquis	0%

Save and close

My Tags

Add a tag and press enter

Predictions

Tag	Probability
maruti suzuki	99.9%
grand marquis	0%
rolls royce	0%

Save and close