

---

# 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

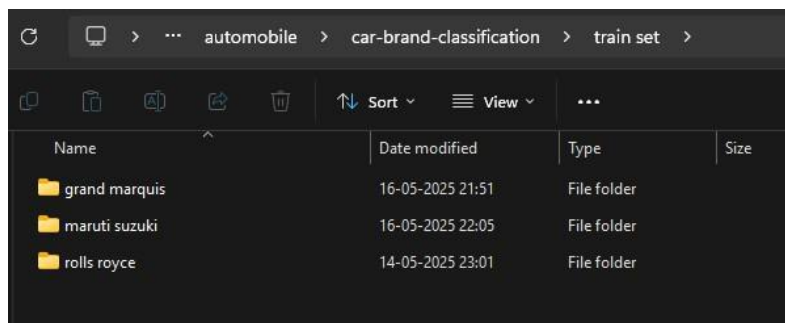
---

I personally have chosen 3 different car brands

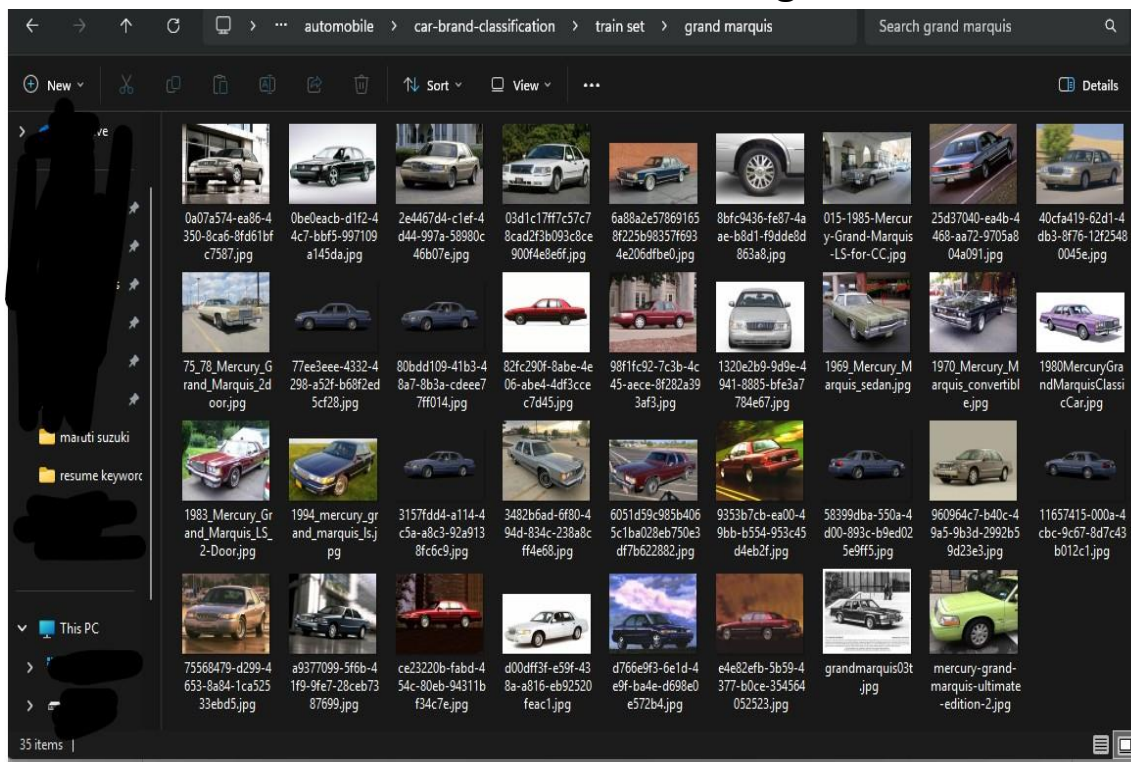
1. Mercury Grand Marquis
2. Maruti Suzuki
3. Rolls Royce

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.



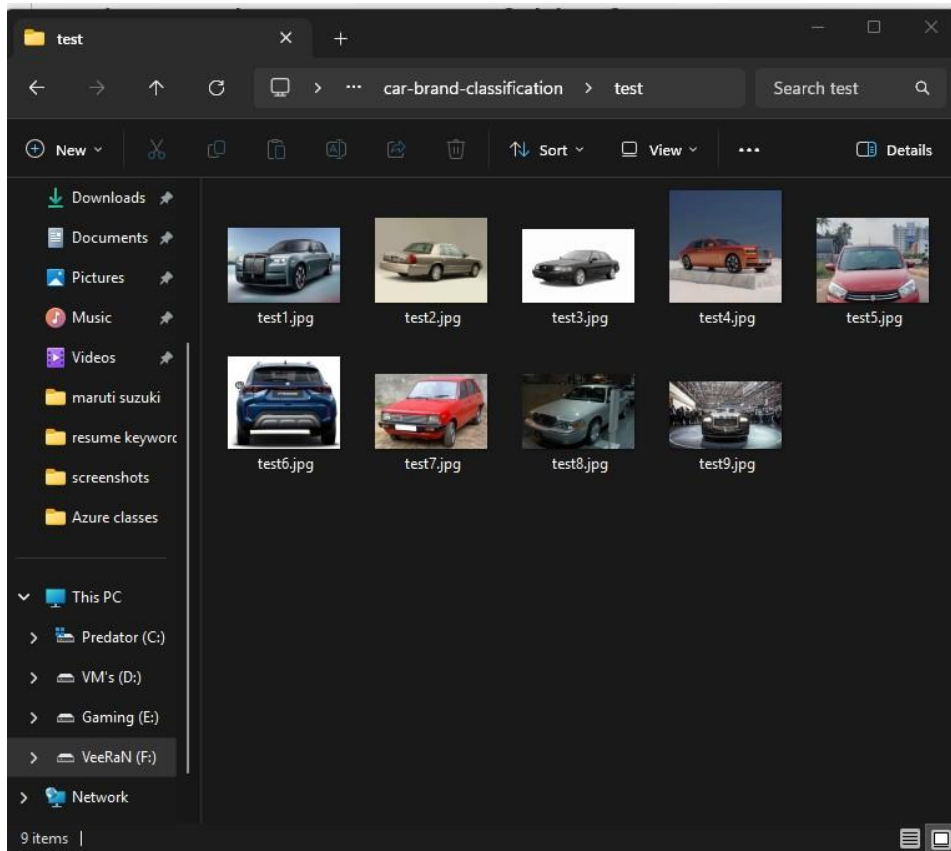
Since each of these folders contain 35 images, each.



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”.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* / Repository name \*  
AshwinAshok3 / car brand classification\_

Your new repository will be created as car-brand-classification .  
The repository name can only contain ASCII letters, digits, and the characters ., -, and \_.

Great repository names are short and memorable. Need inspiration? How about [ideal-chainsaw](#) ?

Description (optional)

☒ Public  
Anyone on the internet can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore  
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license  
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

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.

```
classification> git clone https://github.com/AshwinAshok3/car-brand-classification.git
```

Now type git init

```
ision\automobile\car-brand-classification> 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

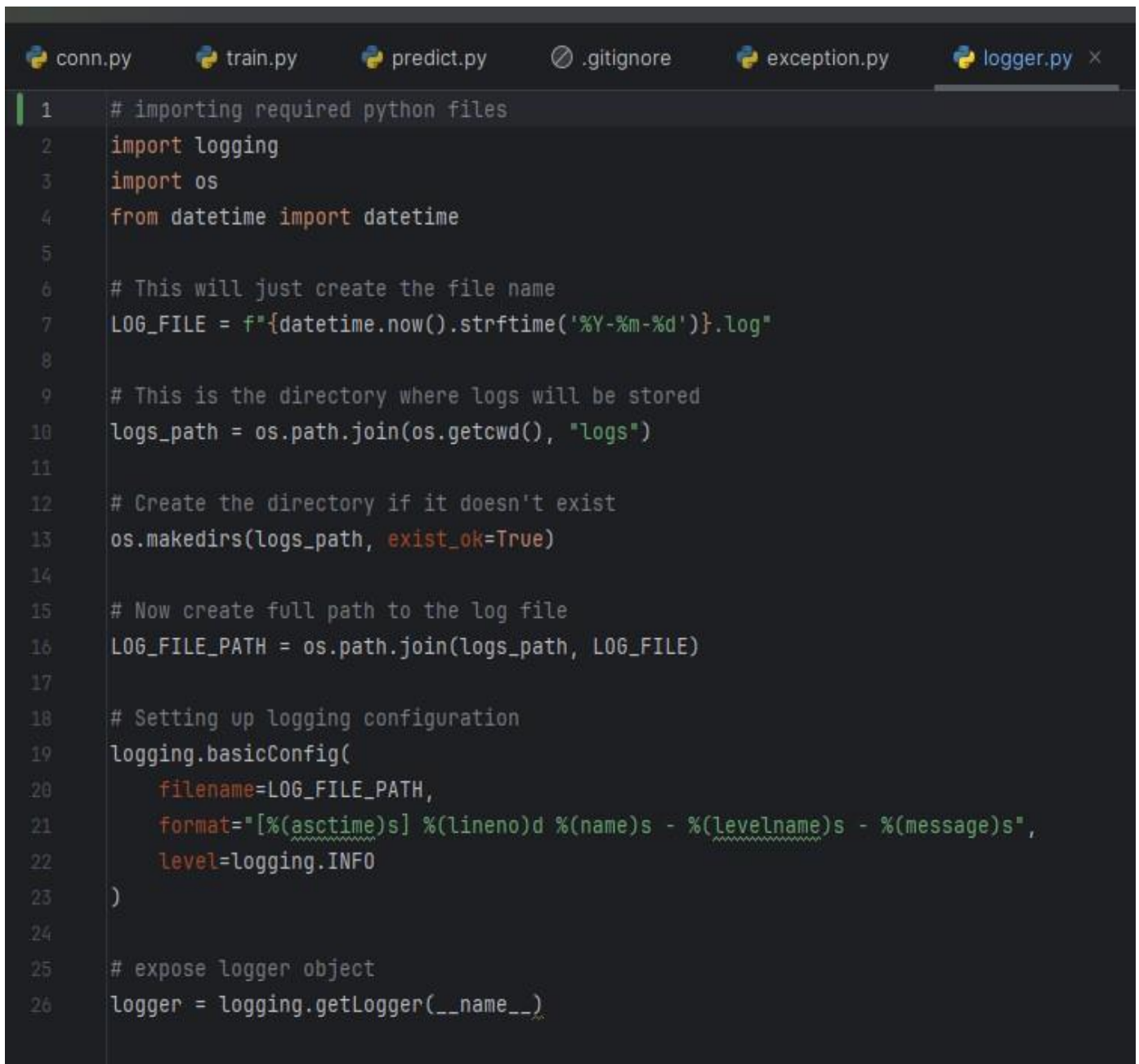
a. ( “pip install azure-cognitiveservices-vision-customvision” )

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:

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'conn.py', 'train.py', 'predict.py', '.gitignore', 'exception.py', and 'logger.py'. The 'logger.py' tab is active. The code in the editor is as follows:

```
1  # importing required python files
2  import logging
3  import os
4  from datetime import datetime
5
6  # This will just create the file name
7  LOG_FILE = f"{datetime.now().strftime('%Y-%m-%d')}.log"
8
9  # This is the directory where logs will be stored
10 logs_path = os.path.join(os.getcwd(), "logs")
11
12 # Create the directory if it doesn't exist
13 os.makedirs(logs_path, exist_ok=True)
14
15 # Now create full path to the log file
16 LOG_FILE_PATH = os.path.join(logs_path, LOG_FILE)
17
18 # Setting up logging configuration
19 logging.basicConfig(
20     filename=LOG_FILE_PATH,
21     format="[%(asctime)s] %(lineno)d %(name)s - %(levelname)s - %(message)s",
22     level=logging.INFO
23 )
24
25 # expose logger object
26 logger = logging.getLogger(__name__)
```



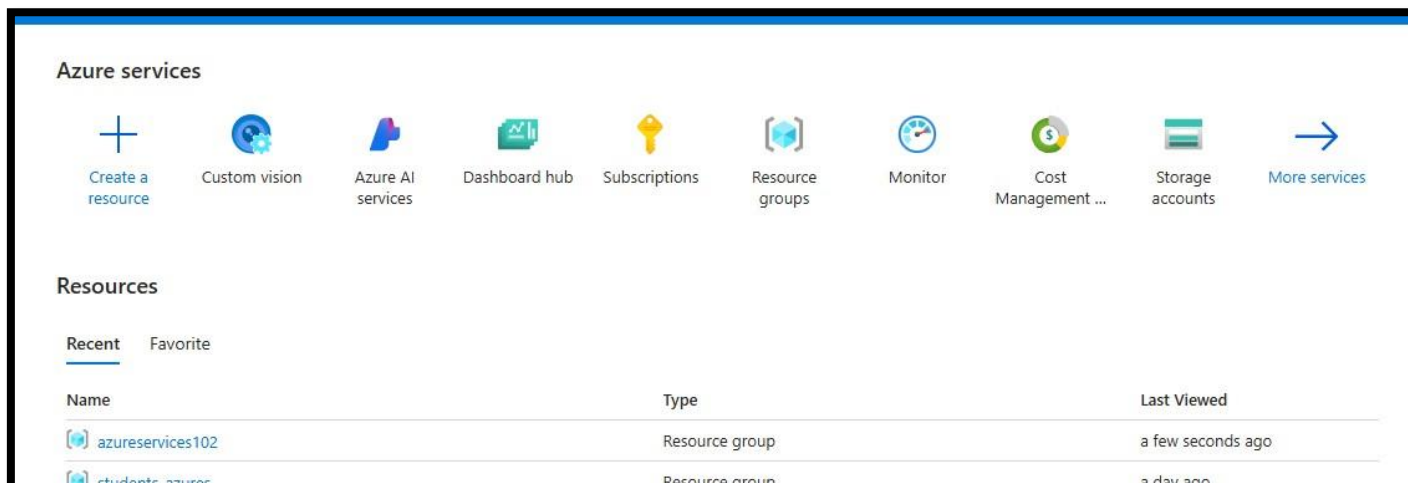
## 6. Now we will define our exceptions (exceptions.py) file:

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

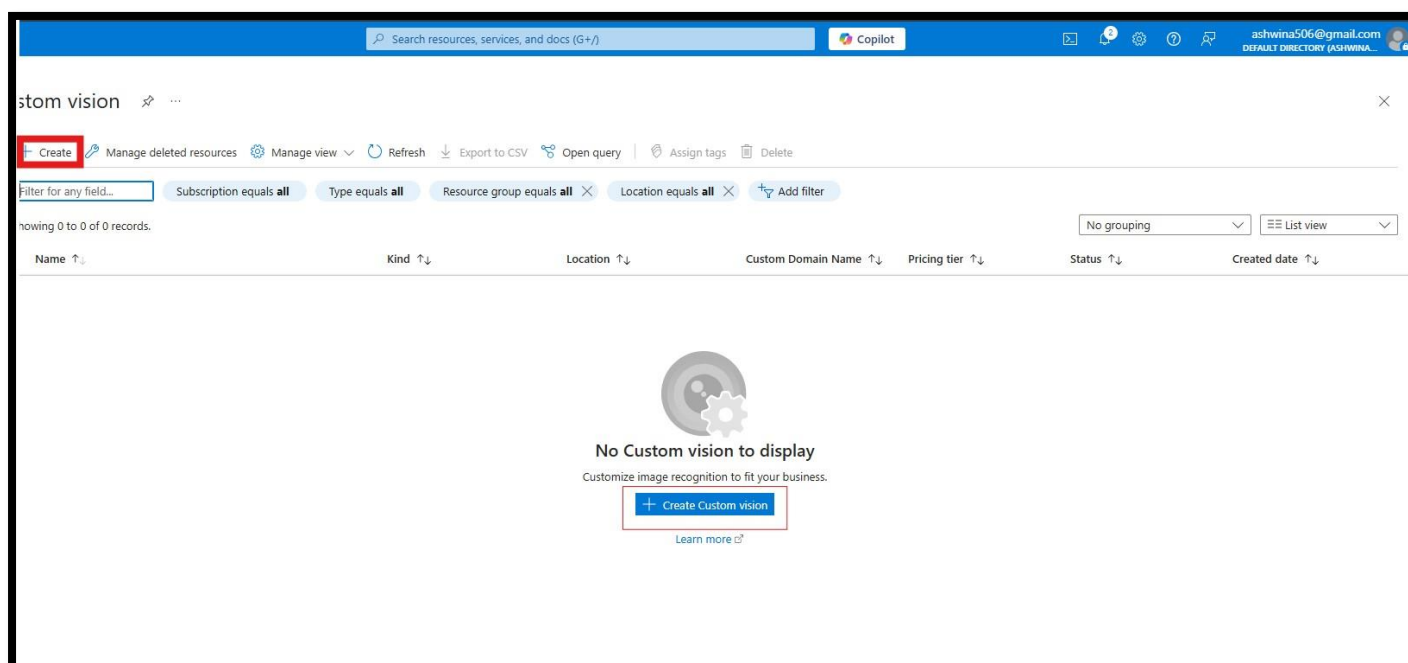
1  # Purpose: Custom Exception Handling with detailed error tracking.
2
3  import sys # To fetch runtime exception details like traceback
4  from logger import logger # Import logger.py
5  # Error message formatting function
6  1 usage  AshwinAshok3
7  def error_message_detail(error, error_detail: sys):
8      __, __, exc_tb = error_detail.exc_info() # Extract traceback info
9      file_name = exc_tb.tb_frame.f_code.co_filename # Which file caused the error
10     line_number = exc_tb.tb_lineno # Line number of error
11
12     # Formatted error message
13     error_message = (
14         f"Error occurred in Python script: [{file_name}], "
15         f"Line Number: [{line_number}], "
16         f"Error Details: [{str(error)}]"
17     )
18     # Log the error
19     logger.error(error_message)
20
21     return error_message
22
23 # Custom Exception Class
24 13 usages  AshwinAshok3
25 class CustomException(Exception):
26     AshwinAshok3
27     def __init__(self, error_message, error_detail: sys):
28         super().__init__(error_message) # Initialize base Exception class
29         self.error_message = error_message_detail(error_message, error_detail)
30
31     AshwinAshok3
32     def __str__(self):
33         return self.error_message # When printed
34
35     AshwinAshok3
36     def __repr__(self):
37         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 .

Home > Azure AI services | Custom vision >

## Create Custom Vision

Basics Network Tags Review + create

Customize and embed state-of-the-art computer vision for specific domains. Build frictionless customer experiences, optimize manufacturing processes, accelerate digital marketing campaigns -- and more. No machine learning expertise is required.

[Learn more](#)

Create options \*

☐ Both  
☐ Prediction  
☒ Training

Project Details

Subscription \* ⓘ Azure for Students

Resource group \* ⓘ azureservices102  
[Create new](#)

Instance Details

Region ⓘ Central India

Name \* ⓘ automobileTraining

Training Resource

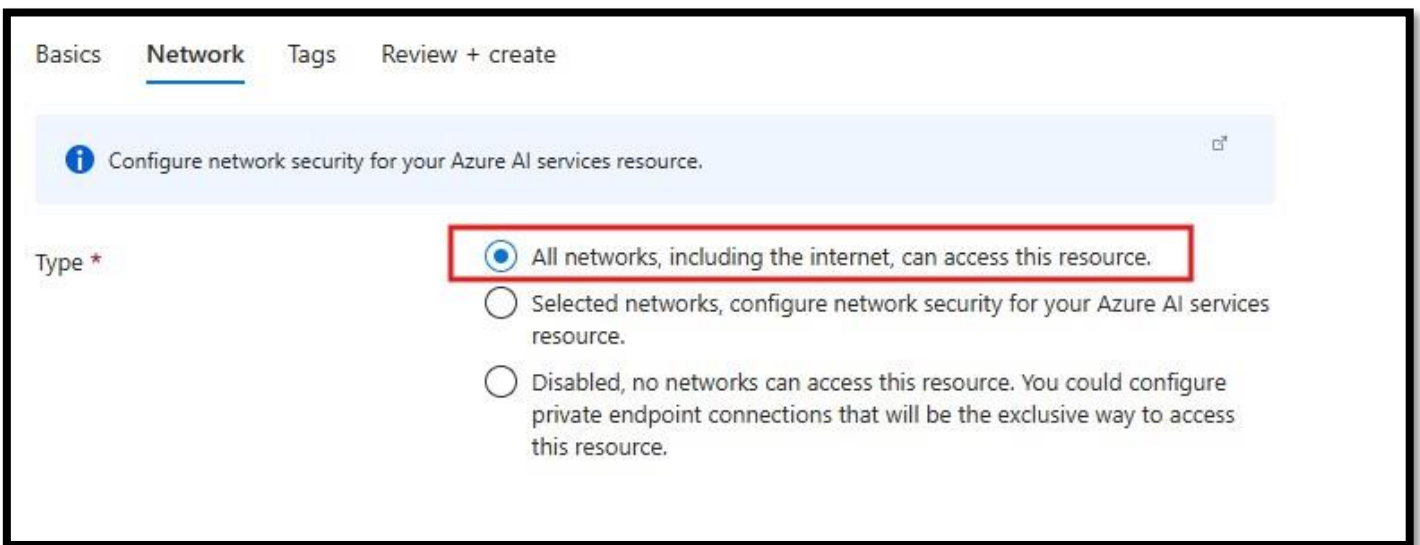
Select pricing for training Resource.

Training pricing tier \* ⓘ Standard S0 (10 Transactions per second)

[View full pricing details](#)

Previous Next Review + create

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 :



The screenshot shows the 'Network' tab of the Azure AI services configuration page. At the top, there are tabs for 'Basics', 'Network', 'Tags', and 'Review + create'. Below the tabs, a light blue banner contains an information icon and the text 'Configure network security for your Azure AI services resource.' with a help icon. Under the 'Type' label, there are three radio button options. The first option, 'All networks, including the internet, can access this resource.', is selected and highlighted with a red rectangle. The other two options are 'Selected networks, configure network security for your Azure AI services resource.' and 'Disabled, no networks can access this resource. You could configure private endpoint connections that will be the exclusive way to access this resource.'

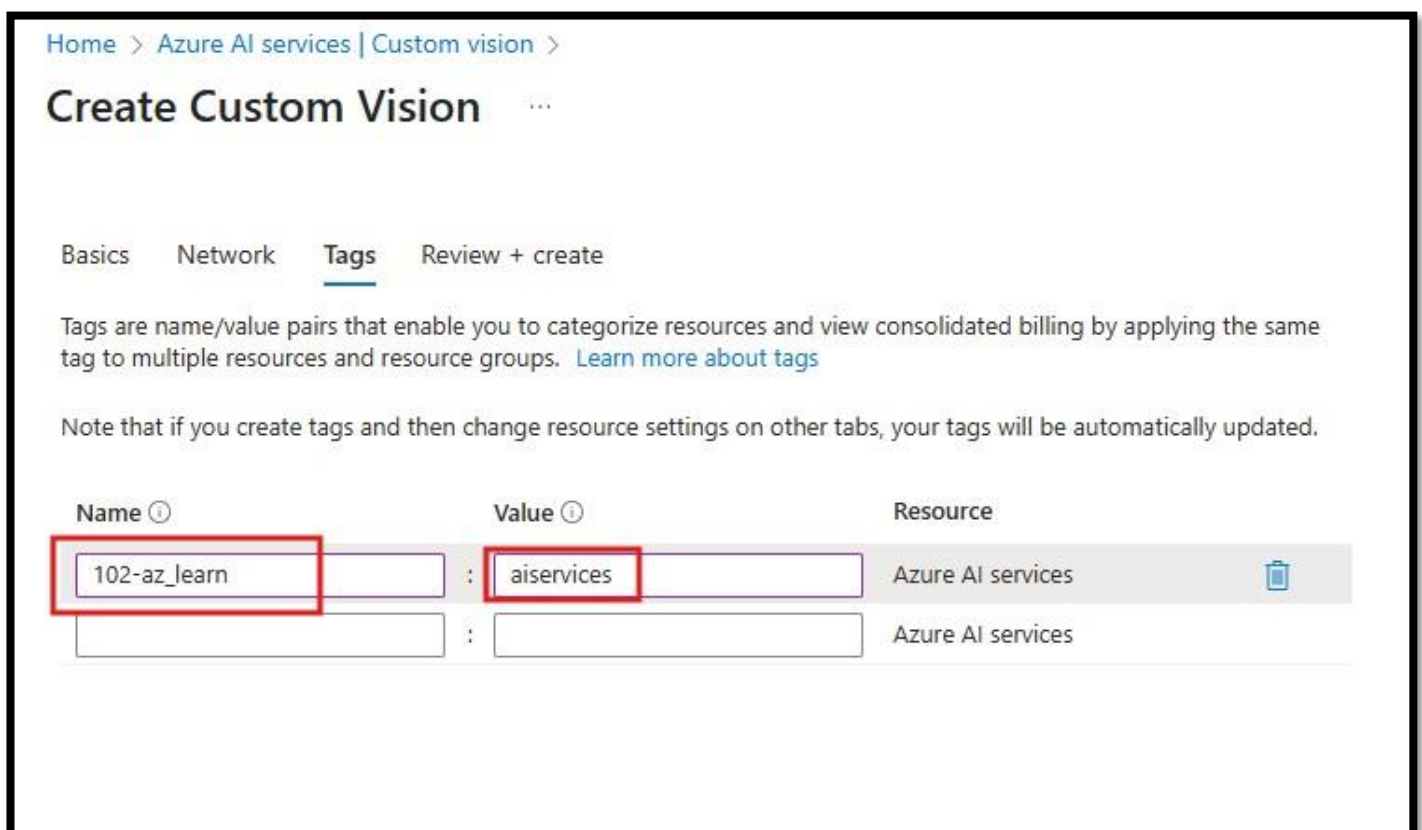
Basics Network Tags Review + create

Configure network security for your Azure AI services resource.

Type \*

- ☒ All networks, including the internet, can access this resource.
- ☐ Selected networks, configure network security for your Azure AI services resource.
- ☐ Disabled, no networks can access this resource. You could configure private endpoint connections that will be the exclusive way to access this resource.

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



The screenshot shows the 'Tags' tab of the 'Create Custom Vision' page. At the top, there are tabs for 'Basics', 'Network', 'Tags', and 'Review + create'. Below the tabs, there is a heading 'Create Custom Vision' followed by an ellipsis. A paragraph explains that tags are name/value pairs used for categorizing resources and consolidated billing, with a link to 'Learn more about tags'. A note states that tags will be automatically updated if resource settings are changed on other tabs. Below this, there is a table with three columns: 'Name', 'Value', and 'Resource'. The first row has '102-az\_learn' in the 'Name' column, 'aiservices' in the 'Value' column, and 'Azure AI services' in the 'Resource' column. The second row has empty input fields for 'Name' and 'Value', and 'Azure AI services' in the 'Resource' column. A red rectangle highlights the '102-az\_learn' text in the first row.

Home > Azure AI services | Custom vision >

## Create Custom Vision ...

Basics Network Tags Review + create

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. [Learn more about tags](#)

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

Name ⓘ	Value ⓘ	Resource
102-az_learn	aiservices	Azure AI services
		Azure AI services

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

Microsoft Azure

Home > Azure AI services | Custom vision >

## Create Custom Vision

Basics Network Tags **Review + create**

[View automation template](#)

### TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the [Azure Marketplace Terms](#) for additional details.

### Basics

Create options	Training
Subscription	Azure for Students
Resource group	azureservices102
Region	Central India
Name	automobileTraining
Training pricing tier	Standard S0 (10 Transactions per second)

### Network

Type	All networks, including the internet, can access this resource.
------	---

### Tags

102-az_learn	aiservices (Azure AI services)
--------------	--------------------------------

[Previous](#) [Next](#) **Create**

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

[Home](#) > [Azure AI services](#) | [Custom vision](#) >

## Create Custom Vision

[Basics](#)   [Network](#)   [Tags](#)   [Review + create](#)

[View automation template](#)

### TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the [Azure Marketplace Terms](#) for additional details.

### Basics

Create options	Prediction
Subscription	Azure for Students
Resource group	azureservices102
Region	Central India
Name	automobilePrediction
Prediction pricing tier	Standard S0 (10K Transactions per month)

### Network

Type	All networks, including the internet, can access this resource.
------	---

### Tags

102-az_learn	aiservices (Azure AI services)
--------------	--------------------------------

[Previous](#)   [Next](#)   [Create](#)



# 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 necessities libraries from python

```
main
Current File
conn.py train.py x predict.py .gitignore exception.py logger.py
1 # Importing necessary modules from Azure SDK and Python standard libraries
2 from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient # Client to access Custom Vision Training API
3 from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateEntry, ImageFileCreateBatch # Models to upload image files in batches
4 from msrest.authentication import ApiKeyCredentials # Handles API key authentication for Azure SDK
5 import os # Provides functions to interact with the operating system (like file and folder navigation)
6 from logger import logger # Custom logger module for logging messages to a file or console
7 from exception import CustomException # Custom exception class for structured error handling
8 import time # Used to delay loop execution (sleep), useful for checking training status repeatedly
9
10
```

## 2. Part II

I hope that you saved your credentials like me

Endpoints | ApiKey\_Training | Resource ID for Prediction

```
10
11 # Saving credentials and model-related configuration
12 endpoint = "https://automobiletraining.cognitiveservices.azure.com/"
13 # Azure endpoint for your Custom Vision resource
14
15 api_key1_training = "FLaCIQL2b83tQjKSqm0sYpTL3pIFQUffh7ciVNIvuPUKQ35gII6MJQQJ998EACGhs1BXJ3w3AAAJAC0G6uNew"
16 # Training key for authentication
17
18 resource_id_prediction = "/subscriptions/7e799c2d-1d5c-44ef-ba1f-9af09461a72c/resourceGroups/azureservices102/providers/Microsoft.CognitiveServices/accounts/automobilePrediction"
19 # Azure resource ID for prediction endpoint
20
21 project_name = "carBrandtraining" # Name of the Custom Vision project to be created
22 publish_iteration_name = "car_brand_model_v1" # Name under which the trained model will be published
23
24
```

### 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
34
```

### 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 be 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.

```

70
71 # Assign tags for each brand (subfolder) and upload images to the project
72 try:
73     for folder in os.listdir(image_folder_path): # Loop through each brand folder
74         folder_path = os.path.join(image_folder_path, folder) # Build full folder path
75
76         if os.path.isdir(folder_path): # Ensure it's a folder
77             tag = trainer.create_tag(project.id, folder) # Create a tag with the folder (brand) name
78             tags_map[folder] = tag # Store tag in dictionary
79             logger.info(f"Tags created for folder {folder}")
80
81         for image_file in os.listdir(folder_path): # Loop through each image in the brand folder
82             image_path = os.path.join(folder_path, image_file) # Full path to image
83             with open(image_path, "rb") as img_data: # Open image in binary mode
84                 try:
85                     logger.info(f"Uploading {image_file}....given tag {folder}....path {image_path}")
86                     entry = ImageFileCreateEntry(
87                         name=image_file, # Name of the image file
88                         contents=img_data.read(), # Read binary content of image
89                         tag_ids=[tag.id] # Assign the image to the corresponding tag
90                     )
91
92                     # Upload the image batch to Azure Custom Vision
93                     upload_result = trainer.create_images_from_files(
94                         project.id,
95                         ImageFileCreateBatch(images=[entry]) # Wrap entry in a batch
96                     )
97
98                     # If upload failed, log error and raise exception
99                     if not upload_result.is_batch_successful:
100                         logger.error(f"Image upload failed for {image_file}. Reasons: {[e.message for e in upload_result.images]}")
101                         raise CustomException(f"Upload failed for {image_file}")
102
103                     logger.info("Image Upload Successful ({^,^})")
104                 except CustomException as e:
105                     logger.error(e)
106                     logger.error("Image Upload Failed ({-,})")
107                     raise CustomException(e) # Propagate exception to main try block
108 except CustomException as e:
109     logger.info("Exception occurred at ", e)
110     logger.error(e)
111     print(e)
112     raise CustomException(e) # Final exception raise if anything failed during upload
113

```

er-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.

```

113
114
115 # Initiate model training
116 print(f"Initiating Training ..")
117 iteration = trainer.train_project(project.id) # Start training on uploaded images
118 logger.info("Training Images initiated....")
119
120
121 # Polling loop: wait for training to complete by checking status every 5 seconds
122 print("Waiting for training to complete...")
123 count = 0
124 while iteration.status != "Completed":
125     count += 1
126     print(f"Training status: {iteration.status}, \niteration count:{count}")
127     time.sleep(5) # Wait 5 seconds before checking status again
128     iteration = trainer.get_iteration(project.id, iteration.id) # Refresh iteration status
129
130
131 # Set this trained iteration as the default version to use for predictions
132 try:
133     logger.info("Training Images. \nProcessing !!!...")
134     trainer.update_iteration(
135         project.id,
136         iteration.id,
137         iteration.name,
138         is_default=True # Set this trained iteration as default
139     )
140     print(f"Training complete for {iteration.id}")
141 except CustomException as e:
142     logger.error(e)
143     raise CustomException(e)
144
145

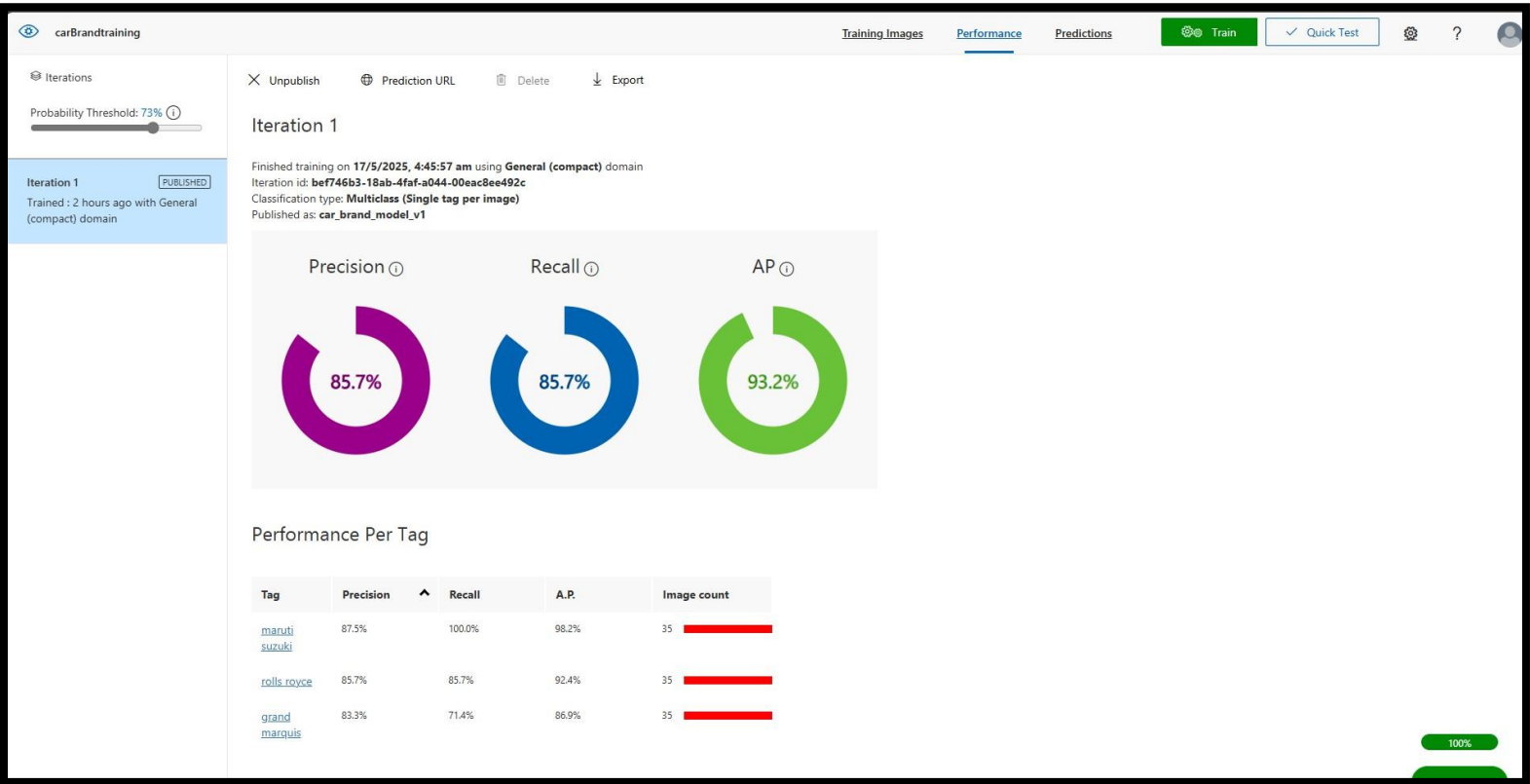
```

## 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.

```
145
146 # Publish the trained model so it can be used for predictions via the Prediction API
147 try:
148     logger.info(f"Instantiating the model : {publish_iteration_name}")
149     trainer.publish_iteration(
150         project_id=project.id,
151         iteration_id=iteration.id,
152         publish_name=publish_iteration_name, # Name to identify the published model
153         prediction_id=resource_id_prediction # Link to prediction endpoint resource
154     )
155     logger.info("Model published successfully ({-,-})")
156 except CustomException as e:
157     logger.info("Model Failed to Publish ({-,-}) !!!")
158     logger.error(e)
159     raise CustomException(e)
160
161 print(f"Model published with name: {publish_iteration_name}")
162 # 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.



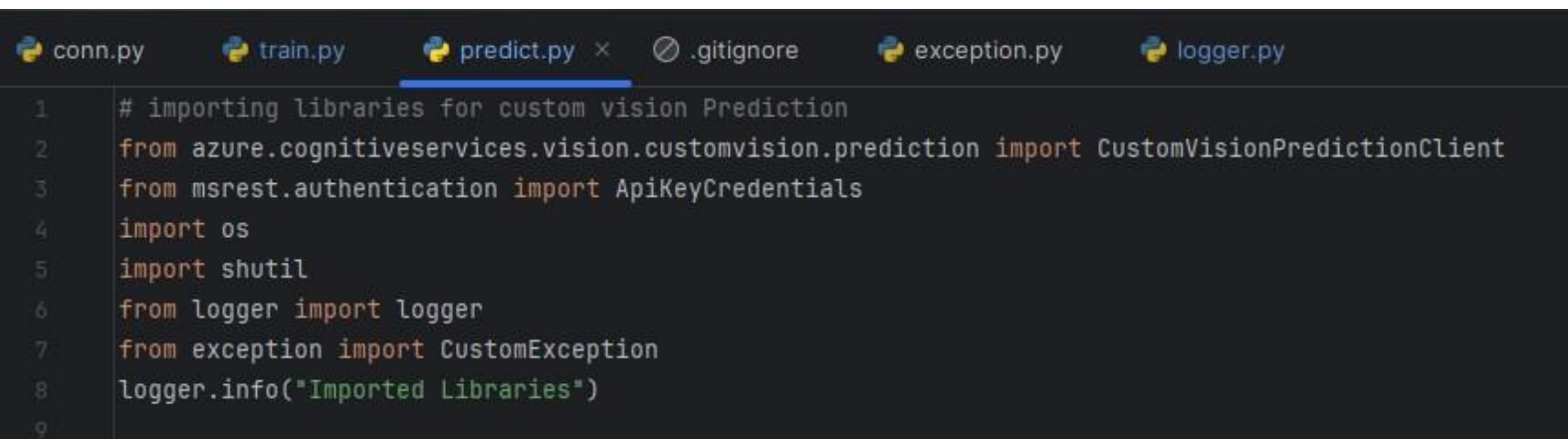
---

# PREDICT.PY

---

In this python file we will deal with prediction related resources same as before we import the same libraries like OS, azure cognitiveservices, 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:



```
conn.py  train.py  predict.py ×  .gitignore  exception.py  logger.py
1  # importing libraries for custom vision Prediction
2  from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
3  from msrest.authentication import ApiKeyCredentials
4  import os
5  import shutil
6  from logger import logger
7  from exception import CustomException
8  logger.info("Imported Libraries")
9
```



## 2nd. Credentials Initializing

- Paths for the output folder and predict data folder as well,
- Ensures if the output directory exists

```
10
11 # Credentials and project details
12 prediction_key = "4EEgJnAYXK1YEouL9UwV2ZoKILN9j3E9FM70D6RrPmcqWbuMuqwkJQQJ99BEAC6hslBXJ3w3AAAIAC0GFTs7"
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

```
28 # Authenticate prediction client
29 try:
30     logger.info("Authentication for Credentials Successful ({{.^}}) /")
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

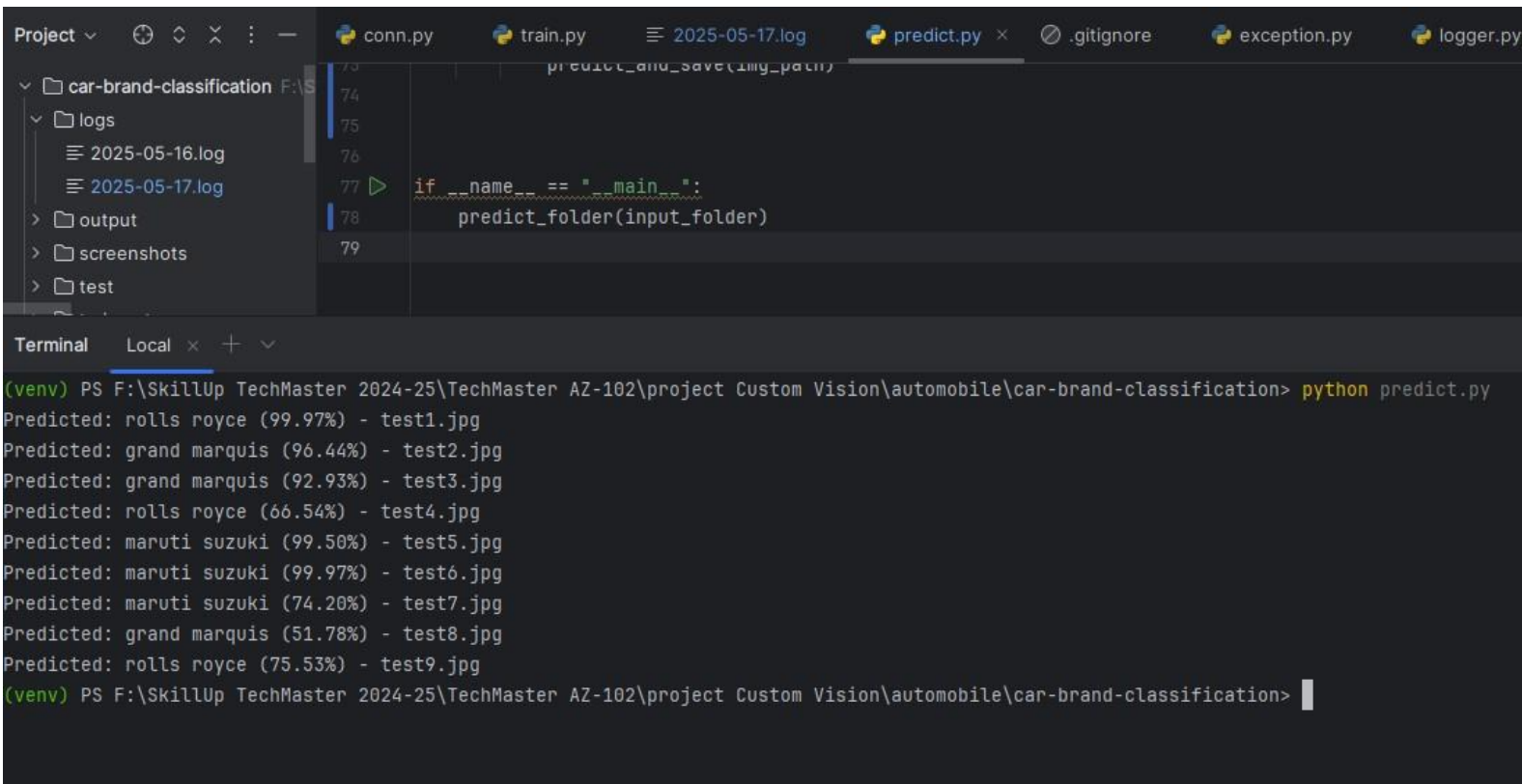
```
39
40 # Batch process folder
41 usage new *
42 def predict_folder(folder_path):
43     for file in os.listdir(folder_path):
44         logger.info("Folder Path Verified")
45         img_path = os.path.join(folder_path, file)
46         if os.path.isfile(img_path):
47             logger.info(f"Image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



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

```
50 # Predict and copy with renamed output
51 1 usage  AshwinAshok3
52 def predict_and_save(image_path):
53     try:
54         logger.info("Reading the Image file ")
55         with open(image_path, "rb") as image_data:
56             results = predictor.classify_image(project_id, published_name, image_data.read())
57     except Exception as e:
58         logger.error(e)
59
60     logger.info("Prep for the prediction")
61     # Get top prediction
62     top_prediction = max(results.predictions, key=lambda x: x.probability)
63     predicted_label = top_prediction.tag_name
64     confidence = top_prediction.probability * 100
65
66     # Print to console
67     print(f"Predicted: {predicted_label} ({confidence:.2f}%) - {os.path.basename(image_path)}")
68
69     # Create new filename and copy to output folder
70     original_name = os.path.basename(image_path)
71     name, ext = os.path.splitext(original_name)
72     new_filename = f"{name}__predictedAS__{predicted_label}{ext}"
73     new_path = os.path.join(output_folder, new_filename)
74
75     shutil.copy(image_path, new_path)
76
77 if __name__ == "__main__":
78     predict_folder(input_folder)
```

# FINAL OUTPUT



The screenshot displays a PyCharm IDE interface. The top pane shows a Python script with the following code:

```
73  
74  
75  
76  
77 if __name__ == "__main__":  
78     predict_folder(input_folder)  
79
```

The bottom pane shows the terminal output of running the script:

```
(venv) PS F:\SkillUp TechMaster 2024-25\TechMaster AZ-102\project Custom Vision\automobile\car-brand-classification> python predict.py  
Predicted: rolls royce (99.97%) - test1.jpg  
Predicted: grand marquis (96.44%) - test2.jpg  
Predicted: grand marquis (92.93%) - test3.jpg  
Predicted: rolls royce (66.54%) - test4.jpg  
Predicted: maruti suzuki (99.50%) - test5.jpg  
Predicted: maruti suzuki (99.97%) - test6.jpg  
Predicted: maruti suzuki (74.20%) - test7.jpg  
Predicted: grand marquis (51.78%) - test8.jpg  
Predicted: rolls royce (75.53%) - test9.jpg  
(venv) PS F:\SkillUp TechMaster 2024-25\TechMaster AZ-102\project Custom Vision\automobile\car-brand-classification>
```

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.



#### 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

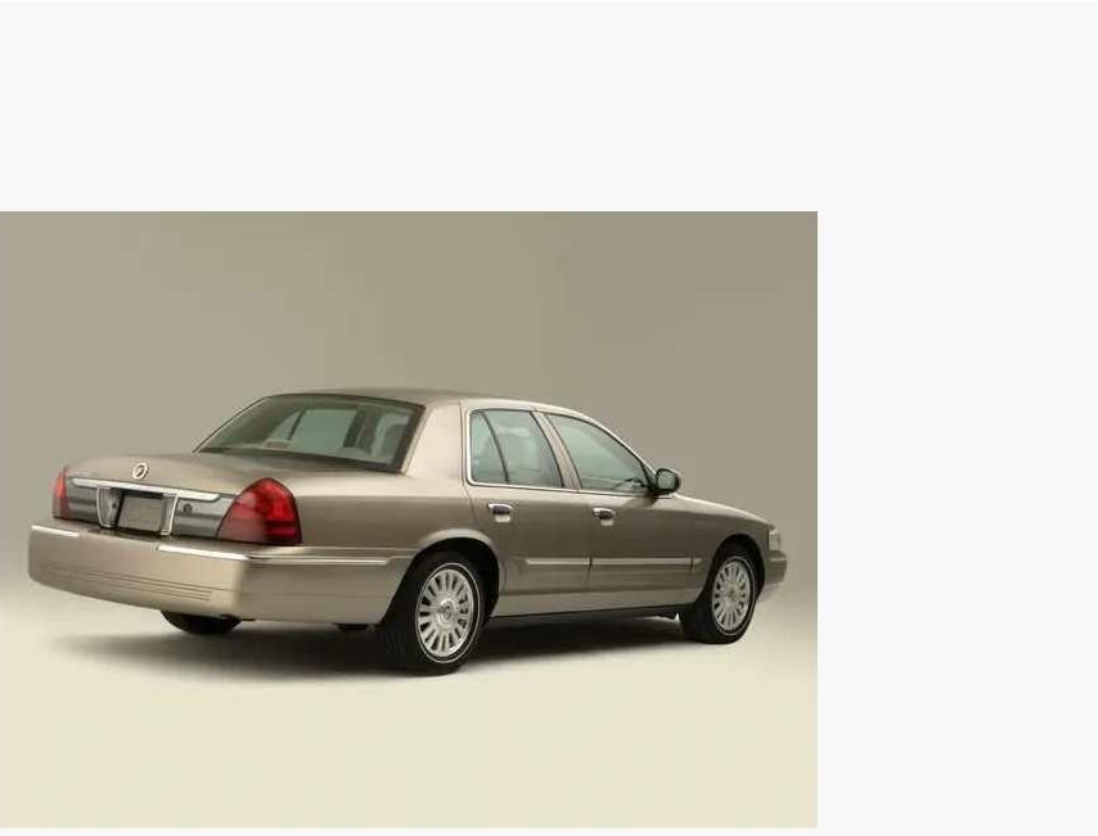




Predictions

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

Save and close



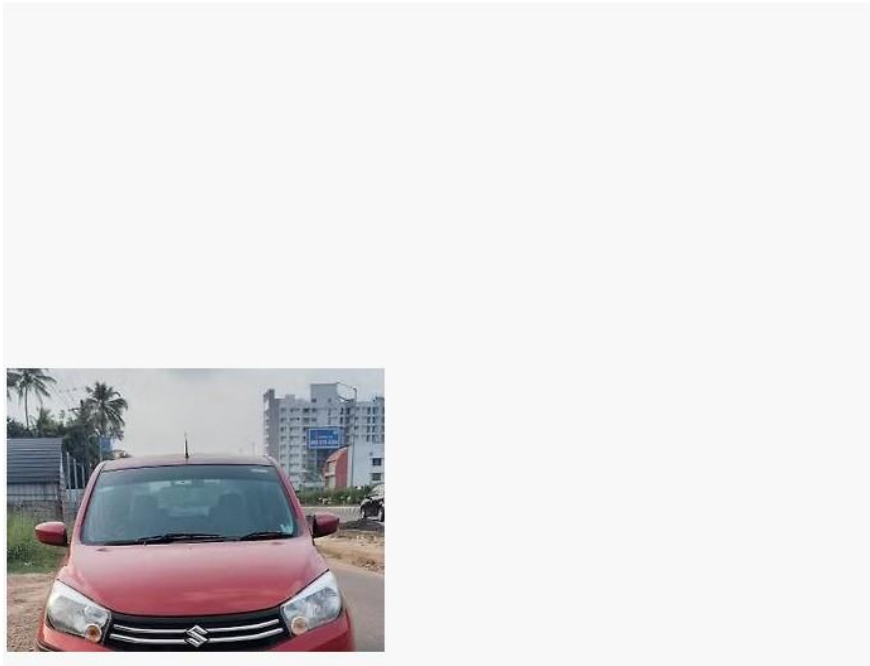
My tags

Add a tag and press enter

Predictions

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

Save and close



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



