Cross validation with deployment	
	from sklearn.datasets import load_iris from sklearn.linear_model import LogisticRegression from sklearn.model_selection import KFold, cross_val_score #We'll also use the cross_val_score function to evaluate our model across the k folds #For demonstration purposes, we'll use the Iris dataset and a simple classifier (LogisticRegression).

	#Load the dataset:
	data = load_iris()
	X, y = data.data, data.target

	data.target
	
	# Set up k-fold cross-validation:
	#We'll use 5-fold cross-validation in this example.
	k = 5
	kf = KFold(n_splits=k, shuffle=True, random_state=43) #Random state here because it want wheneve you run it gives same data points in that 5 box
	#n_splits=k specifies the number of folds.
	#shuffle=True ensures that the data is shuffled before splitting into folds.
	#random_state=42 is an arbitrary seed for reproducibility.

	#Apply k-fold cross-validation using a classifier:
	
	#We'll use the cross_val_score function to evaluate the performance of our LogisticRegression classifier across the k folds.
	clf = LogisticRegression(max_iter=200, random_state=42)
	
	##max_iter=200: This specifies the maximum number of iterations the solver (i.e., the algorithm used to find the logistic regression coefficients) will take to converge. The default value is 100, but sometimes, increasing this number can help in cases where the solver might not converge quickly.
	#Max_iter- This parameter is used when you call the fit method on a logistic regression model to train it on a specific dataset.
	
	#Performing Cross-Validation:
	scores = cross_val_score(clf, X, y, cv=kf)
	
	#cv=kf: This specifies the cross-validation splitting strategy to be used. In our case, we're using the kf object we created earlier, which is a 5-fold cross-validation strategy with shuffling.

	#scores will contain the accuracy of the classifier for each of the k folds. scores #1. means 100% accuracy

	#Analyze the results:
	
	#We can compute the average and standard deviation of the scores to get an idea of the model's performance and variability across the folds.
	average_accuracy = scores.mean()
	std_accuracy = scores.std()
	
	#Approximately 97.33%
	average_accuracy
	
	#Standard Deviation of Accuracy: Approximately 2.49% (it means 2.49% var from mean of all accuracy)
	std_accuracy

	#Note #1. Cross validation do fit the model and discard- the purpose of this not to train model and use for prediction - the purpose of this to find accuray of differene - new data #2.If you want to train model and use for prediction then you need to fit again and do prediction

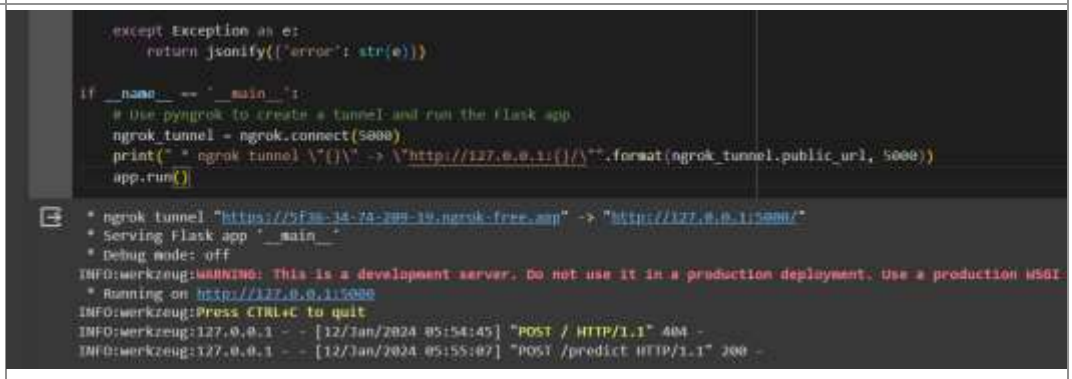
	#Supose if you want to train model for prediction
	clf = LogisticRegression(max_iter=200, random_state=42)
	clf.fit(X, y)

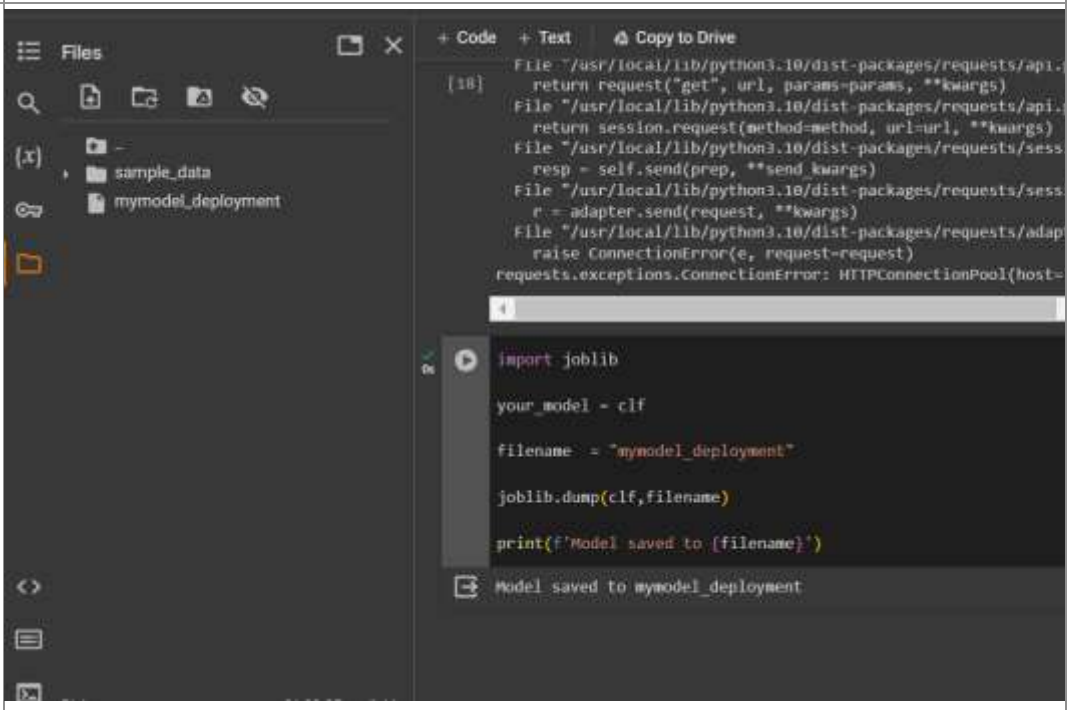
	#Now, you can use this trained model (clf) to make predictions on new data. Let's say you have some random data new_data:
	new_data = [[5.0, 3.5, 1.5, 0.2]]
	predicted_class = clf.predict(new_data)

	predicted_class_value = predicted_class[0] predicted_class_value #0 means setosa
	#Install flask ngrok, to expose this enviroment publicly !pip install pyngrok

```
from flask import Flask, request, jsonify
from pyngrok import ngrok
import numpy as np
import joblib

# Set your Ngrok authentication token
ngrok.set_auth_token('2aqBTjLoJNULDJEaUNJD5hWnNkc_4hi26fzaejcQ3UjQhWaNB')

app = Flask(__name__)

# Load pre-trained model (replace 'your_model.pkl' with the actual model file)
model = clf

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Mapping of class labels to flower names
        class_names = {0: "setosa", 1: "versicolor", 2: "virginica"}

        data = request.json
        prediction = model.predict(np.array([data['input']]).reshape(1, -1))
        flower_name = class_names[int(prediction[0])]

        return jsonify({'prediction': int(prediction[0]), 'flower_name': flower_name})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    # Use pyngrok to create a tunnel and run the Flask app
    ngrok_tunnel = ngrok.connect(5000)
    print(" * ngrok tunnel \"{}\" ->
\"http://127.0.0.1:{}/\"".format(ngrok_tunnel.public_url, 5000))
    app.run()
```

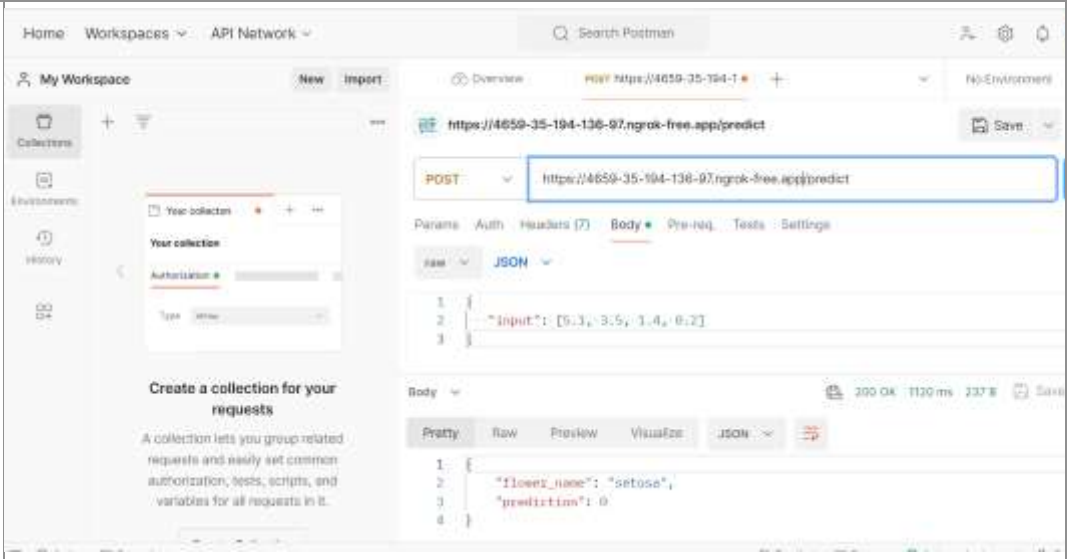
```
#To stop server
ngrok.kill()
```



save the model and deploy it	
	import joblib your_model = clf filename = "mymodel_deployment" joblib.dump(clf,filename) print(f'Model saved to {filename}')
File is Created	

First Upload the File in New Machine and Run the Code ; Before that add the AuthToken in Code and file name	# prompt: No module named 'pyngrok'
	!pip install pyngrok
	
	from flask import Flask, request, jsonify
	from pyngrok import ngrok
	import numpy as np
	import joblib
	
	# Set your Ngrok authentication token
	ngrok.set_auth_token('2aqNEGlmK2gUAsDcf2qQNLFV3qS_6ETQCYPCXv8PC6cxM7GnX')
	
	app = Flask(__name__)
	
	# Load pre-trained model (replace 'your_model.pkl' with the actual model file)
	filename = 'mymodel_deployment'
	
	# Load the model from the file
	model = joblib.load(filename)

	@app.route('/predict', methods=['POST'])
	def predict():
	  try:
	  # Mapping of class labels to flower names
	  class_names = {0: "setosa", 1: "versicolor", 2: "virginica"}
	
	  data = request.json
	  prediction = model.predict(np.array([data['input']]).reshape(1, -1))
	  flower_name = class_names[int(prediction[0])]
	
	  return jsonify({'prediction': int(prediction[0]), 'flower_name': flower_name})
	
	  except Exception as e:
	  return jsonify({'error': str(e)})
	
	if __name__ == '__main__':
	  # Use pyngrok to create a tunnel and run the Flask app
	  ngrok_tunnel = ngrok.connect(5000)
	  print(" * ngrok tunnel \"{}\" ->\"http://127.0.0.1:{}/\"".format(ngrok_tunnel.public_url, 5000))
	  app.run()

Copy the ngrok tunnal URL and past in HTTP request	
Select Request type as HTTP and past ngrok tunnal URL In Text Field, select parameter style as raw and in body Add input	
Load the model	
Created empty variable	Db[]

Create two api's i.e predict & update	
Store the data in model(train)	
Predict	
Update(if it predict wrong)	
Check accuracy	