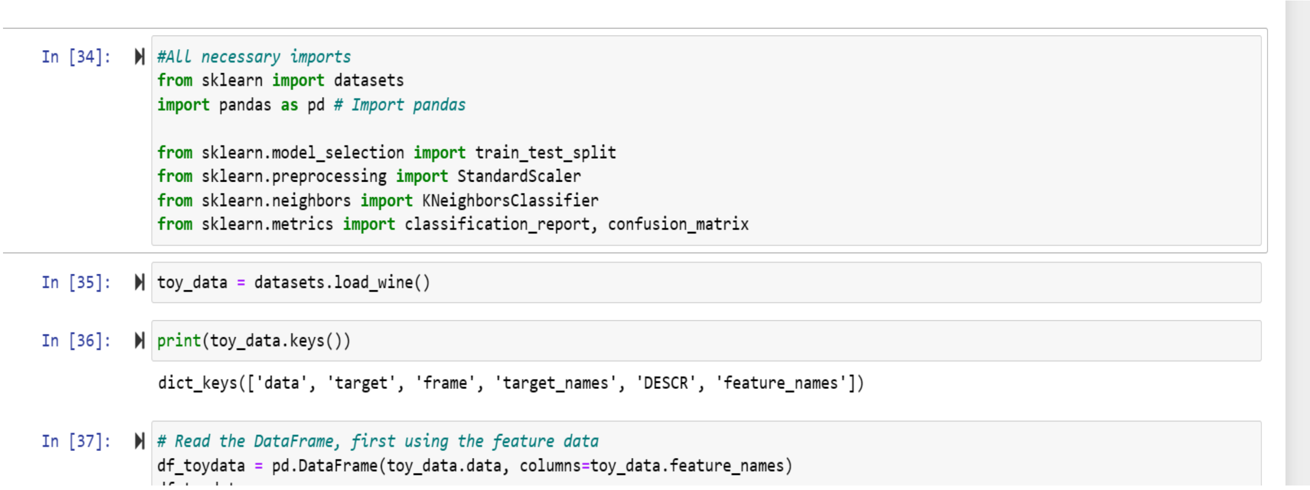
Toy Data Deployment on Flask

For this activity, we will be using some toy data available on sklearn to build the model for a classification problem. We will use wine data and classify them into their type. We will perform the following steps:

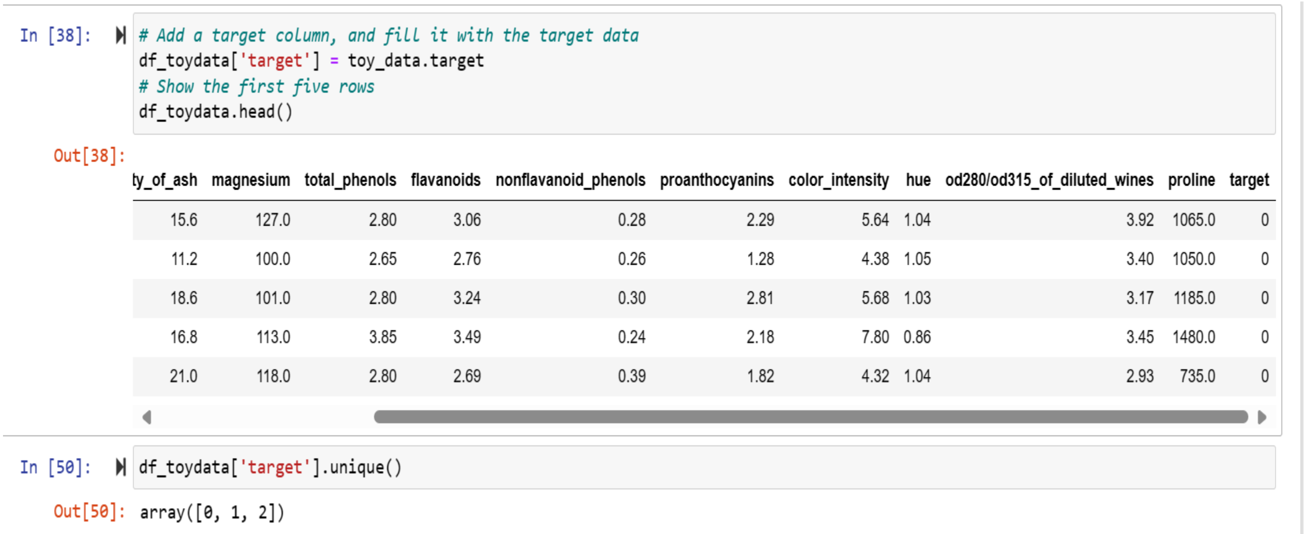
1. Create a machine learning model.
2. Save the model using Pickle.
3. Develop a web application with Flask and integrate the model in the application.
4. Deploy the model on Flask.

1. Create a machine learning model.

We will load the wine recognition data from sklearn. The dataset contains 13 numeric features and target class. Target column has 3 different values 0, 1, 2. Based on 13 features of the wine data we will classify the wine into these three categories.



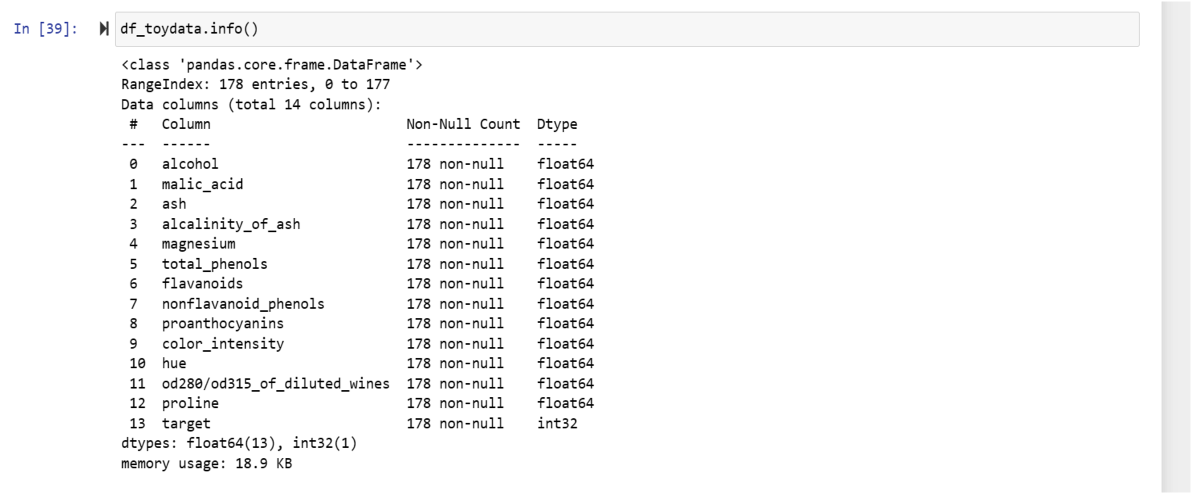
```
In [34]:  #All necessary imports
          from sklearn import datasets
          import pandas as pd # Import pandas

          from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import StandardScaler
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import classification_report, confusion_matrix

In [35]:  toy_data = datasets.load_wine()

In [36]:  print(toy_data.keys())

          dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])

In [37]:  # Read the DataFrame, first using the feature data
          df_toydata = pd.DataFrame(toy_data.data, columns=toy_data.feature_names)
```



```
In [38]:  # Add a target column, and fill it with the target data
          df_toydata['target'] = toy_data.target
          # Show the first five rows
          df_toydata.head()

Out[38]:
```

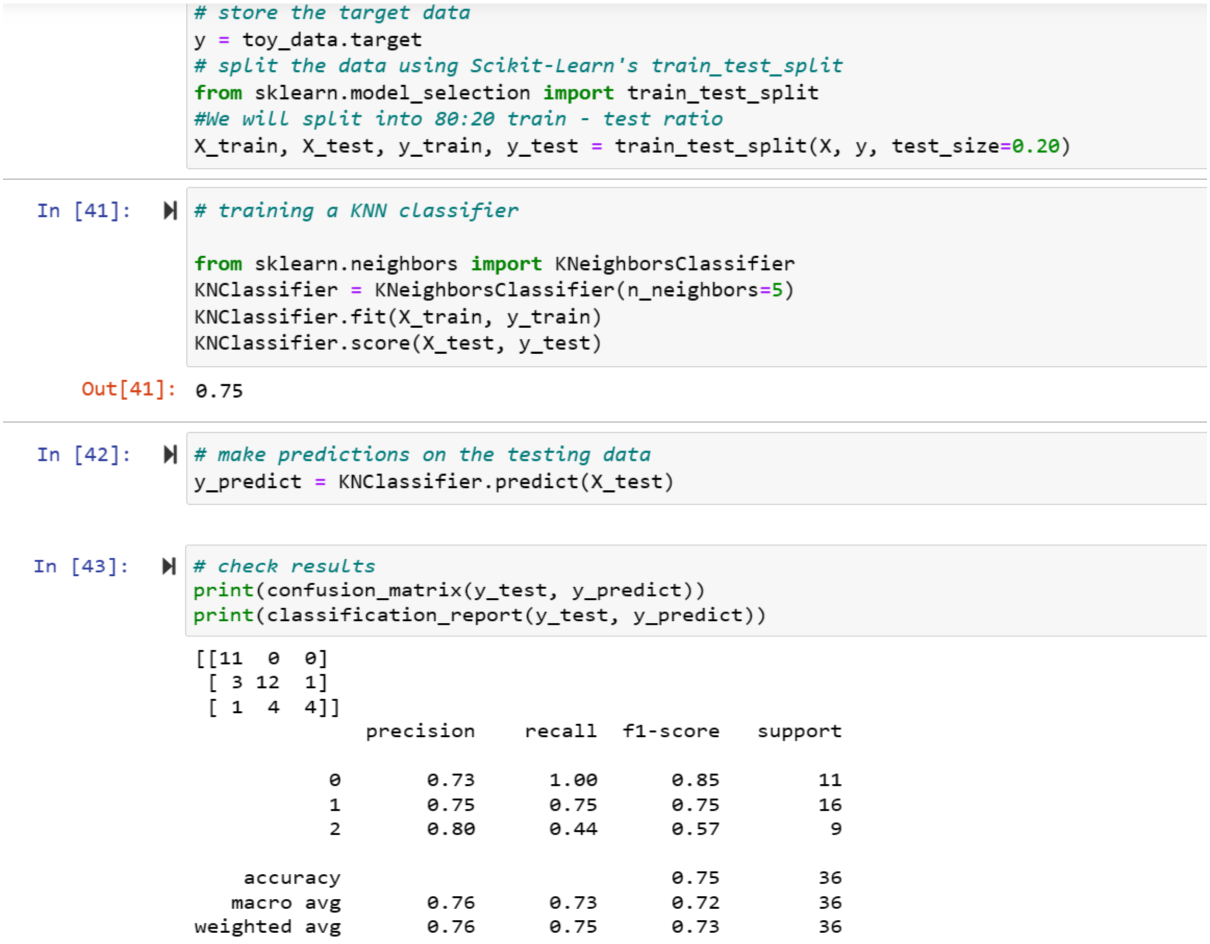
ty_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

```
In [50]:  df_toydata['target'].unique()

Out[50]:  array([0, 1, 2])
```



```
In [39]:  df_toydata.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int32  
dtypes: float64(13), int32(1)
memory usage: 18.9 KB
```

We will use in-built model K-Neighbor Classifier. From the data available, we will split the data into train and test dataset as 80:20 ratio.



```
# store the target data
y = toy_data.target
# split the data using Scikit-Learn's train_test_split
from sklearn.model_selection import train_test_split
#We will split into 80:20 train - test ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
In [41]:  # training a KNN classifier

          from sklearn.neighbors import KNeighborsClassifier
          KNClassifier = KNeighborsClassifier(n_neighbors=5)
          KNClassifier.fit(X_train, y_train)
          KNClassifier.score(X_test, y_test)

Out[41]:  0.75
```

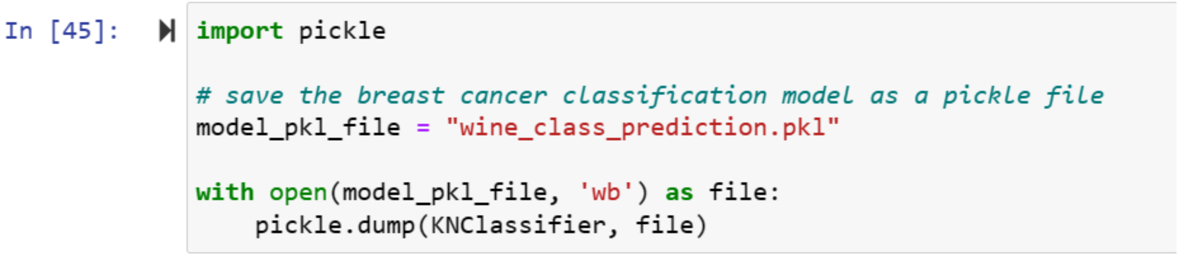
```
In [42]:  # make predictions on the testing data
          y_predict = KNClassifier.predict(X_test)
```

```
In [43]:  # check results
          print(confusion_matrix(y_test, y_predict))
          print(classification_report(y_test, y_predict))

          [[11  0  0]
           [ 3 12  1]
           [ 1  4  4]]
                        precision    recall  f1-score   support

                     0       0.73      1.00      0.85        11
                     1       0.75      0.75      0.75        16
                     2       0.80      0.44      0.57         9

              accuracy                           0.75        36
             macro avg       0.76      0.73      0.72        36
          weighted avg       0.76      0.75      0.73        36
```

Now, we will save the ML model for future use. We will save the model by using the pickle file.

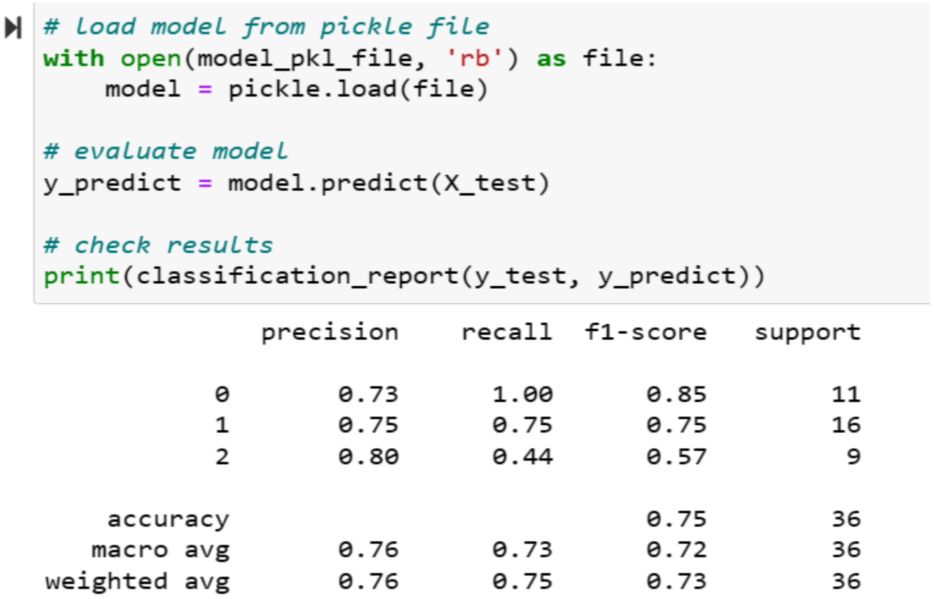
2. Save the model using Pickle file.

We need to save the model to deploy it and to be able to use it later with some other inputs. We will save our pretrained model using pickle using the following code:



```
In [45]:  import pickle

          # save the breast cancer classification model as a pickle file
          model_pkl_file = "wine_class_prediction.pkl"

          with open(model_pkl_file, 'wb') as file:
              pickle.dump(KNClassifier, file)
```

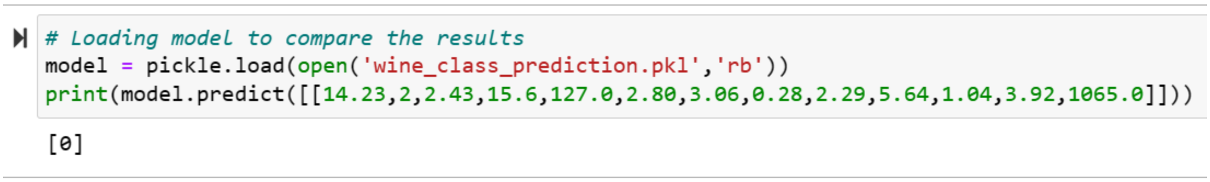
We will write in binary mode (wb) from the pretrained model called KNClassifier and store it in a pickle file called, “wine_class_prediction.pkl”. The dump() method stores the model in the given pickle file. Now, we will open the file in rb (read binary) mode to load the saved model.
We will now load the model from the pickle file and make predictions. Below is the scores.



```
# load model from pickle file
with open(model_pkl_file, 'rb') as file:
    model = pickle.load(file)

# evaluate model
y_predict = model.predict(X_test)

# check results
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.73      1.00      0.85        11
           1       0.75      0.75      0.75        16
           2       0.80      0.44      0.57         9

    accuracy                           0.75        36
   macro avg       0.76      0.73      0.72        36
weighted avg       0.76      0.75      0.73        36
```

We would test one data from the input data we have so to see if the model is predicting any right value.



```
# Loading model to compare the results
model = pickle.load(open('wine_class_prediction.pkl','rb'))
print(model.predict([[14.23,2,2.43,15.6,127.0,2.80,3.06,0.28,2.29,5.64,1.04,3.92,1065.0]]))
```

```
[0]
```

Since, our model has now been built and it can now predict the wine category, we will prepare the model towards its deployment.

3. Develop a web application with Flask and integrate the model in the application.

Before going further we would create the following files.

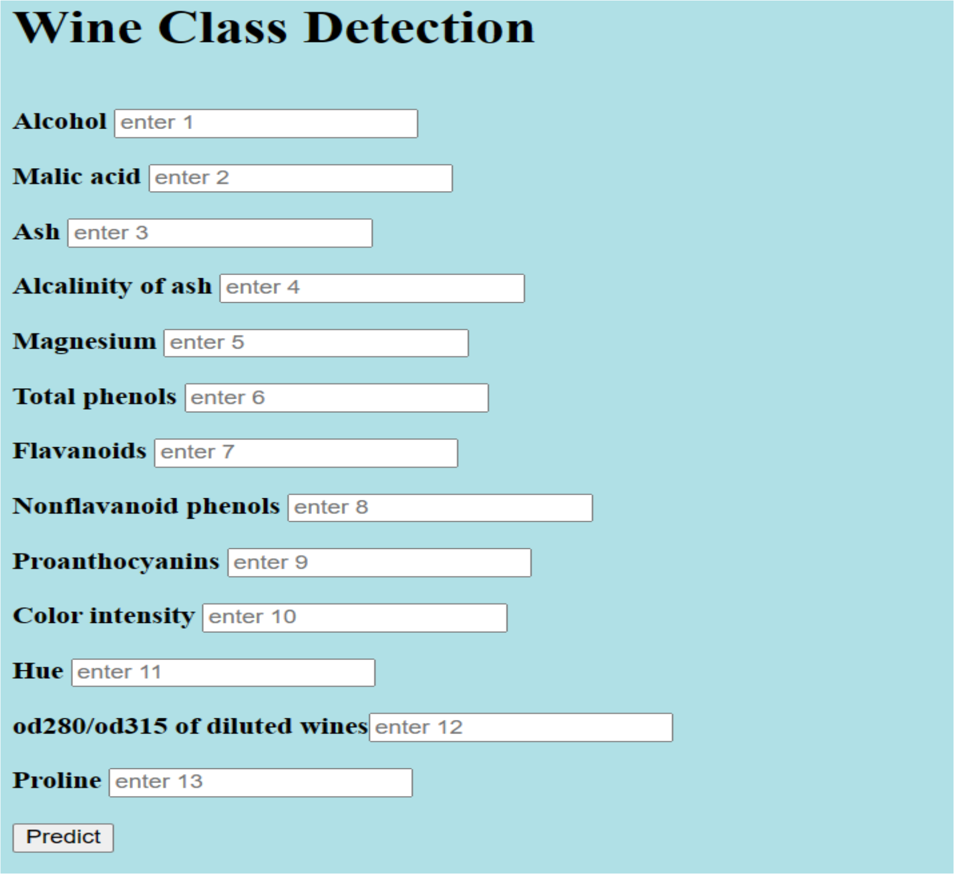
a. An HTML file (home.html)
b. Create a python file (app.py)

a. An HTML file:

We will create a webpage that will ask the user to provide all those 13 input features as input and will display the target, i.e. category of the wine, based on the feature values provided.

```
<!DOCTYPE html>
<html>
<body style="background-color:powderblue">
        <h1>Wine Class Detection</h1><br>
<form action="{{url_for('predict')}}", method="POST">
<b>
        Alcohol <input type="text", name='a', placeholder="enter 1"><br><br>
        Malic acid <input type="text", name='b', placeholder="enter 2"><br><br>
        Ash <input type="text", name='c', placeholder="enter 3"><br><br>
        Alcalinity of ash <input type="text", name='d', placeholder="enter 4"><br><br>
        Magnesium <input type="text", name='e', placeholder="enter 5"><br><br>
        Total phenols <input type="text", name='f', placeholder="enter 6"><br><br>
        Flavanoids <input type="text", name='g', placeholder="enter 7"><br><br>
        Nonflavanoid phenols <input type="text", name='h', placeholder="enter 8"><br><br>
        Proanthocyanins <input type="text", name='i', placeholder="enter 9"><br><br>
        Color intensity <input type="text", name='j', placeholder="enter 10"><br><br>
        Hue <input type="text", name='k', placeholder="enter 11"><br><br>
        od280/od315 of diluted wines<input type="text", name='l', placeholder="enter 12"><br><br>
        Proline  <input type="text", name='m', placeholder="enter 13"><br><br>

        <button type="submit", class="btn">Predict</button><br><br>

</form>
<!--Prediction Result-->
        <div id ="result">
                <strong style="color:red">{{prediction_text}}</strong>
        </div>
</body>
</html>
```

The webpage would look something like this.



Wine Class Detection

Alcohol enter 1

Malic acid enter 2

Ash enter 3

Alcalinity of ash enter 4

Magnesium enter 5

Total phenols enter 6

Flavanoids enter 7

Nonflavanoid phenols enter 8

Proanthocyanins enter 9

Color intensity enter 10

Hue enter 11

od280/od315 of diluted wines enter 12

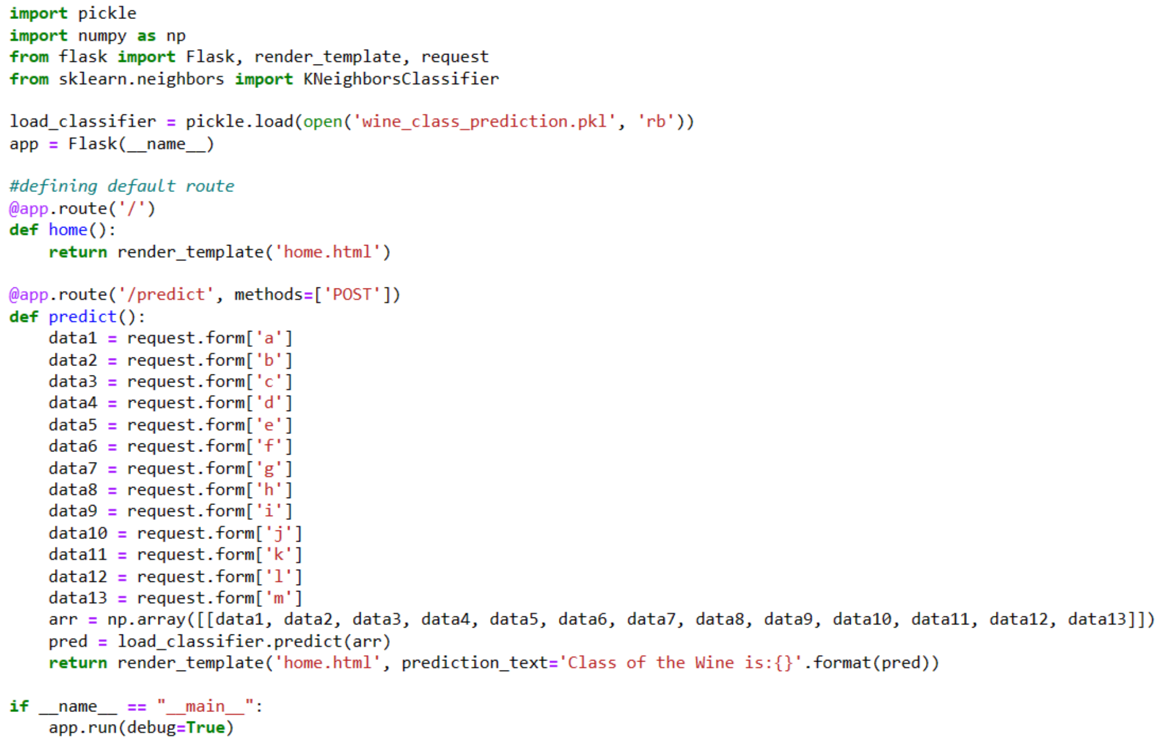
Proline enter 13

b. Create a Python file (app.py)

In this python file, we will define the operations for execution of the model or application we have built. It will load the load pickle file created earlier during the model building stage and then we will run the code on Flask. This file will take all the data entered by the user on the webpage and apply the pretrained classifier on the data and will predict and display the category of the wine on screen.

From the code below, we can see we have first installed all the necessary imports. As part of which is to import Flask library, render_template, and request. We then created a Flask instance and assigned it to a variable called app. Also, we created some URL routes using @app.route(), which would correspond to various web pages of our application.

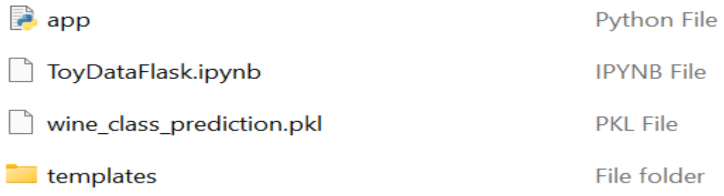
app.run will start the server and will load the application on the web browser.



```
import pickle
import numpy as np
from flask import Flask, render_template, request
from sklearn.neighbors import KNeighborsClassifier

load_classifier = pickle.load(open('wine_class_prediction.pkl', 'rb'))
app = Flask(__name__)

#defining default route
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['POST'])
def predict():
    data1 = request.form['a']
    data2 = request.form['b']
    data3 = request.form['c']
    data4 = request.form['d']
    data5 = request.form['e']
    data6 = request.form['f']
    data7 = request.form['g']
    data8 = request.form['h']
    data9 = request.form['i']
    data10 = request.form['j']
    data11 = request.form['k']
    data12 = request.form['l']
    data13 = request.form['m']
    arr = np.array([[data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, data12, data13]])
    pred = load_classifier.predict(arr)
    return render_template('home.html', prediction_text='Class of the Wine is:{}'.format(pred))

if __name__ == "__main__":
    app.run(debug=True)
```

4. Deploy the model on Flask.

We would first create a folder structure. We need to make sure we have a separate directory for all the files related to the current project and will move or delete the files non-relevant to the project. We need to create a templates folder inside the working directory as we need to store the html file inside the templates folder. The working directory would now contain a pickle file (for the pretrained model), a python file (in this case, app.py) and a templates directory.

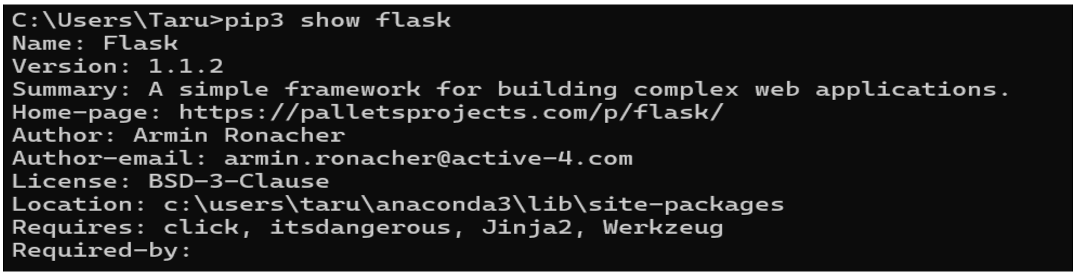


app	Python File
ToyDataFlask.ipynb	IPYNB File
wine_class_prediction.pkl	PKL File
templates	File folder

Flask can be installed using the command below:

```
pip3 install flask
```

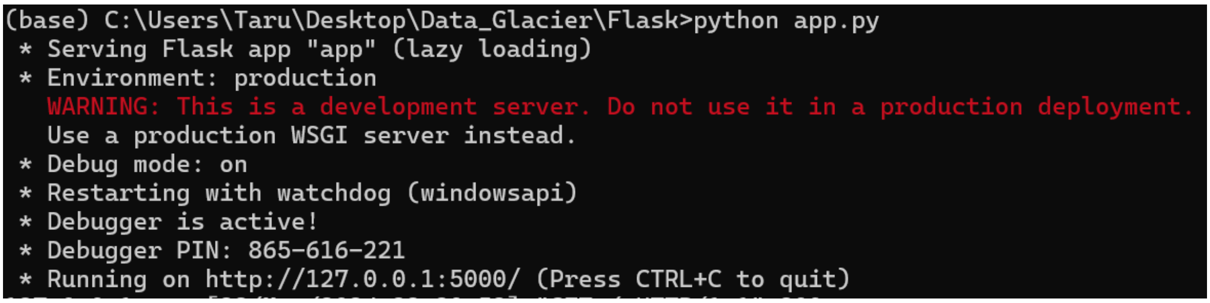
However, I already had flask installed. We can check the version of the flask installed using the following command on command prompt.



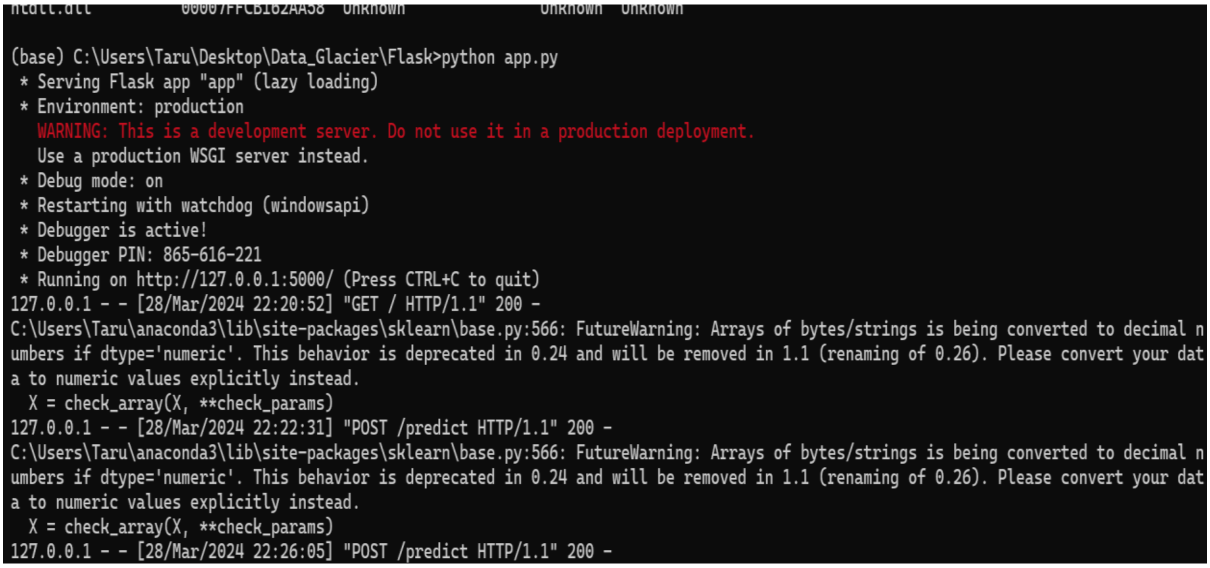
```
C:\Users\Taru>pip3 show flask
Name: Flask
Version: 1.1.2
Summary: A simple framework for building complex web applications.
Home-page: https://palletsprojects.com/p/flask/
Author: Armin Ronacher
Author-email: armin.ronacher@active-4.com
License: BSD-3-Clause
Location: c:\users\taru\anaconda3\lib\site-packages
Requires: click, itsdangerous, Jinja2, Werkzeug
Required-by:
```

Being a Windows user, as I have anaconda installed on my computer, I used Anaconda shell to run and deploy the application on Flask. On the shell, we need to go to the directory where the Python file is present using "cd <path>". Then, we run the command python <filename.py> ("python app.py") on shell.

The output is as shown below:

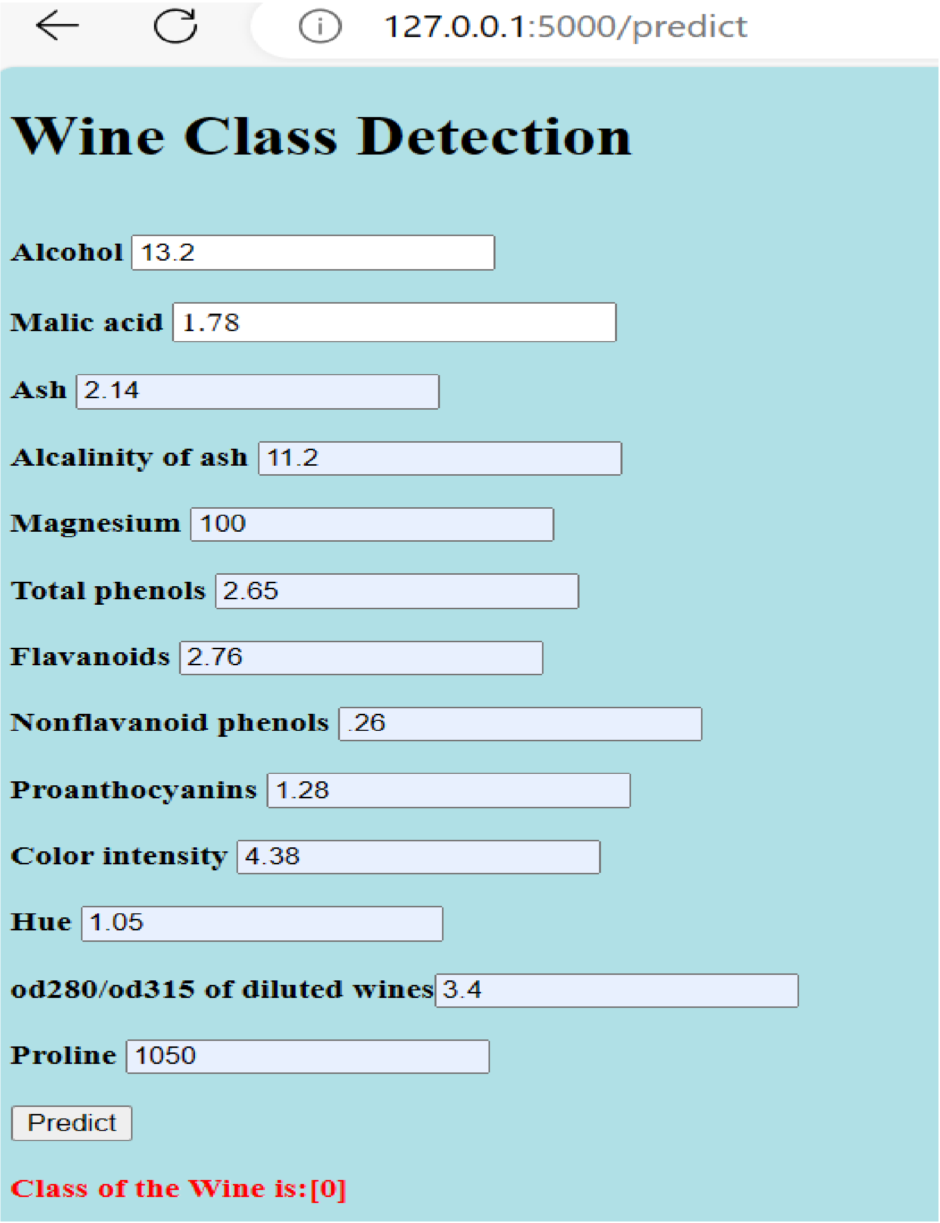


```
(base) C:\Users\Taru\Desktop\Data_Glacier\Flask>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 865-616-221
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



```
ntdll.dll          00007FFCB162AA58  Unknown               Unknown  Unknown

(base) C:\Users\Taru\Desktop\Data_Glacier\Flask>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 865-616-221
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [28/Mar/2024 22:20:52] "GET / HTTP/1.1" 200 -
C:\Users\Taru\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal n
umbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your dat
a to numeric values explicitly instead.
  X = check_array(X, **check_params)
127.0.0.1 - - [28/Mar/2024 22:22:31] "POST /predict HTTP/1.1" 200 -
C:\Users\Taru\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal n
umbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your dat
a to numeric values explicitly instead.
  X = check_array(X, **check_params)
127.0.0.1 - - [28/Mar/2024 22:26:05] "POST /predict HTTP/1.1" 200 -
```

When we clicked on the url http://127.0.0.1:5000/, it displayed the webpage we created using HTML file. When we entered the values, it predicted the wine category as shown below:



127.0.0.1:5000/predict

Wine Class Detection

Alcohol 13.2

Malic acid 1.78

Ash 2.14

Alcalinity of ash 11.2

Magnesium 100

Total phenols 2.65

Flavanoids 2.76

Nonflavanoid phenols .26

Proanthocyanins 1.28

Color intensity 4.38

Hue 1.05

od280/od315 of diluted wines 3.4

Proline 1050

Class of the Wine is:[0]

In this article, we saw an example of deployment with Flask.