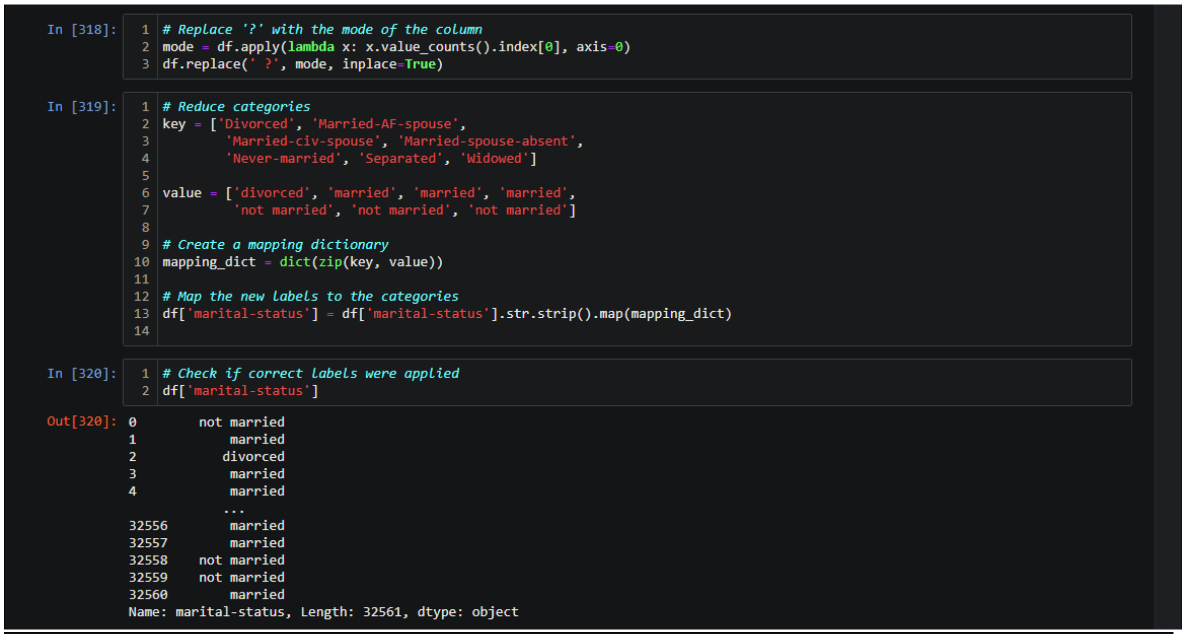
Andrew O’Drain

Lisum24

8-28-2023

FLASK DEPLOYMENT: DECISION TREE CLASSIFIER DEPLOYED ON FLASK

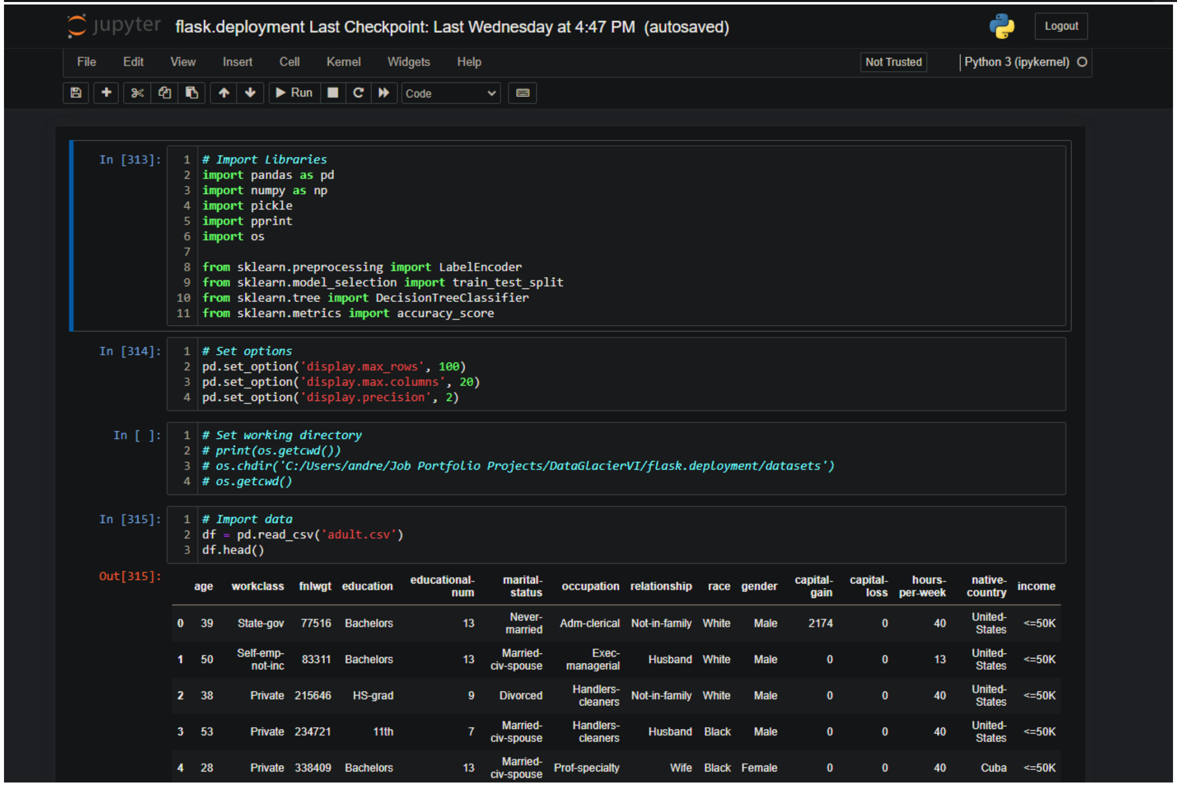


```
In [318]:
# Replace '?' with the mode of the column
mode = df.apply(lambda x: x.value_counts().index[0], axis=0)
df.replace(' ?', mode, inplace=True)
```

```
In [319]:
# Reduce categories
key = ['Divorced', 'Married-AF-spouse',
        'Married-civ-spouse', 'Married-spouse-absent',
        'Never-married', 'Separated', 'Widowed']

value = ['divorced', 'married', 'married', 'married',
         'not married', 'not married', 'not married']

# Create a mapping dictionary
mapping_dict = dict(zip(key, value))

# Map the new labels to the categories
df['marital-status'] = df['marital-status'].str.strip().map(mapping_dict)

```

```
In [320]:
# Check if correct labels were applied
df['marital-status']
```

```
Out[320]:
0        not married
1            married
2           divorced
3            married
4            married
            ...
32556        married
32557        married
32558    not married
32559    not married
32560        married
Name: marital-status, Length: 32561, dtype: object
```



```
In [313]:
# Import Libraries
import pandas as pd
import numpy as np
import pickle
import pprint
import os

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

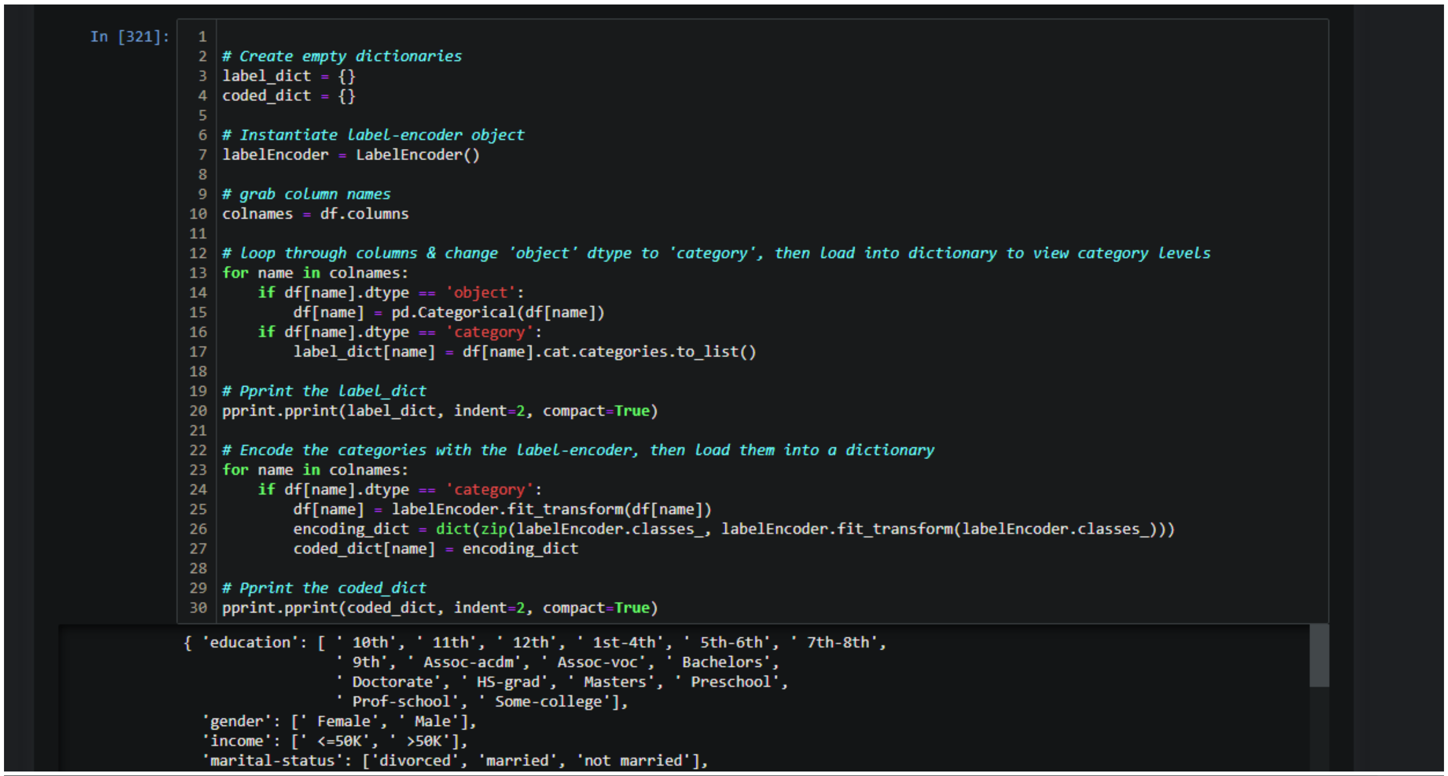
```
In [314]:
# Set options
pd.set_option('display.max_rows', 100)
pd.set_option('display.max.columns', 20)
pd.set_option('display.precision', 2)
```

```
In [ ]:
# Set working directory
# print(os.getcwd())
# os.chdir('C:/Users/andre/Job Portfolio Projects/DataGlacierVI/flask.deployment/datasets')
# os.getcwd()
```

```
In [315]:
# Import data
df = pd.read_csv('adult.csv')
df.head()
```

Out[315]:

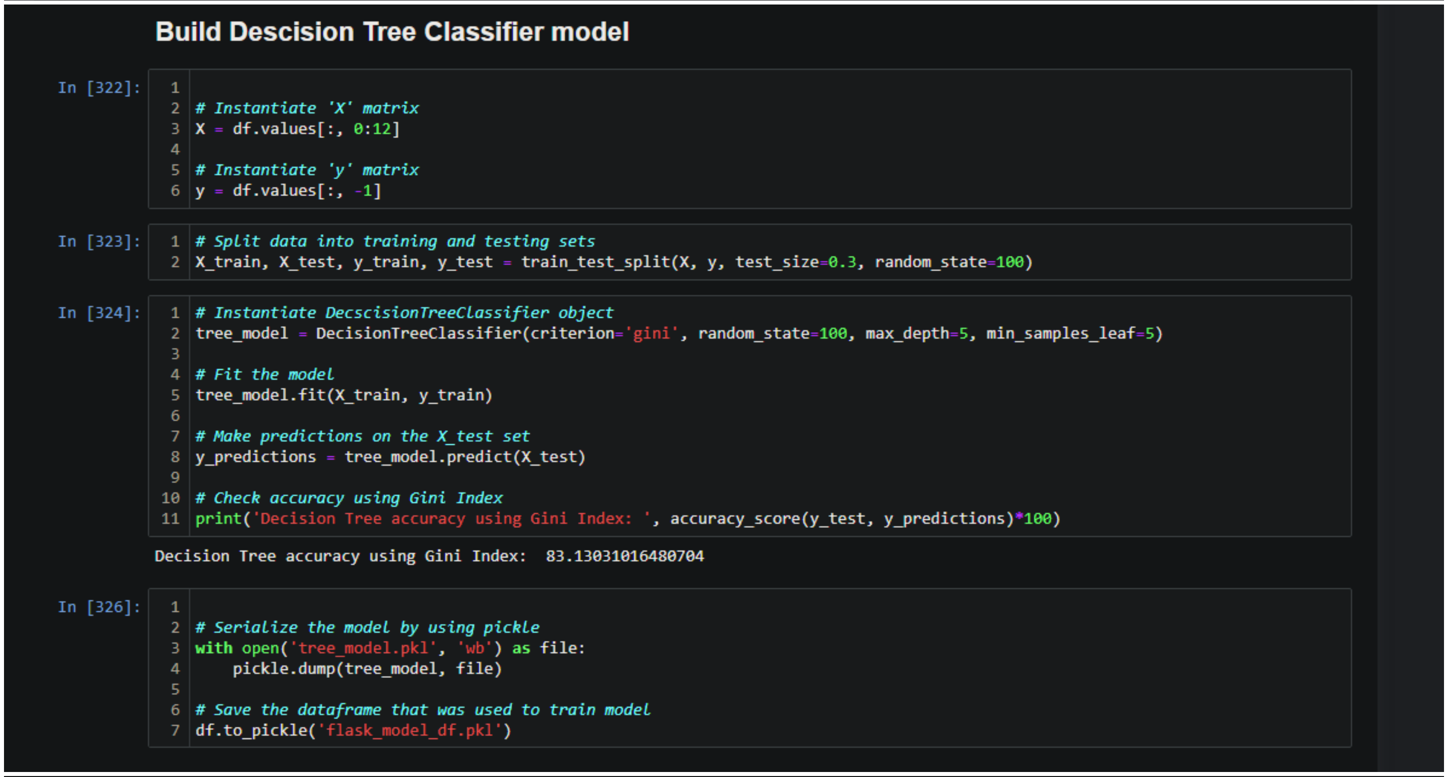
	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K



In [321]:

```python

# Create empty dictionaries
label_dict = {}
coded_dict = {}

# Instantiate label-encoder object
labelEncoder = LabelEncoder()

# grab column names
colnames = df.columns

# loop through columns & change 'object' dtype to 'category', then load into dictionary to view category levels
for name in colnames:
    if df[name].dtype == 'object':
        df[name] = pd.Categorical(df[name])
    if df[name].dtype == 'category':
        label_dict[name] = df[name].cat.categories.to_list()

# Pprint the label_dict
pprint.pprint(label_dict, indent=2, compact=True)

# Encode the categories with the label-encoder, then load them into a dictionary
for name in colnames:
    if df[name].dtype == 'category':
        df[name] = labelEncoder.fit_transform(df[name])
        encoding_dict = dict(zip(labelEncoder.classes_, labelEncoder.fit_transform(labelEncoder.classes_)))
        coded_dict[name] = encoding_dict

# Pprint the coded_dict
pprint.pprint(coded_dict, indent=2, compact=True)
```

```
{ 'education': [ ' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',
                 ' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors',
                 ' Doctorate', ' HS-grad', ' Masters', ' Preschool',
                 ' Prof-school', ' Some-college'],
  'gender': [' Female', ' Male'],
  'income': [' <=50K', ' >50K'],
  'marital-status': ['divorced', 'married', 'not married'],
```



Build Descision Tree Classifier model

In [322]:

```python

# Instantiate 'X' matrix
X = df.values[:, 0:12]

# Instantiate 'y' matrix
y = df.values[:, -1]
```

In [323]:

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100)
```

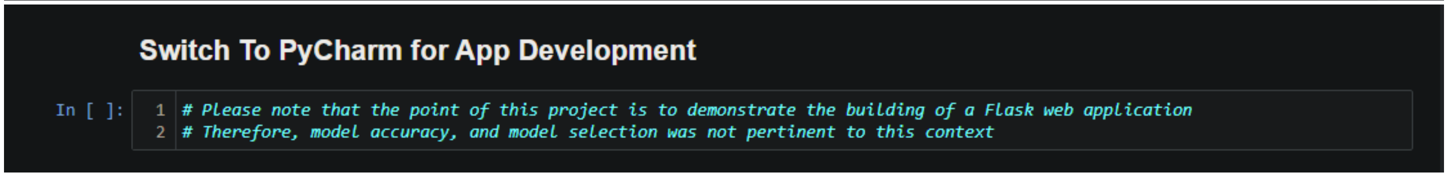
In [324]:

```python
# Instantiate DecscisionTreeClassifier object
tree_model = DecisionTreeClassifier(criterion='gini', random_state=100, max_depth=5, min_samples_leaf=5)

# Fit the model
tree_model.fit(X_train, y_train)

# Make predictions on the X_test set
y_predictions = tree_model.predict(X_test)

# Check accuracy using Gini Index
print('Decision Tree accuracy using Gini Index: ', accuracy_score(y_test, y_predictions)*100)
```

```
Decision Tree accuracy using Gini Index:  83.13031016480704
```

In [326]:

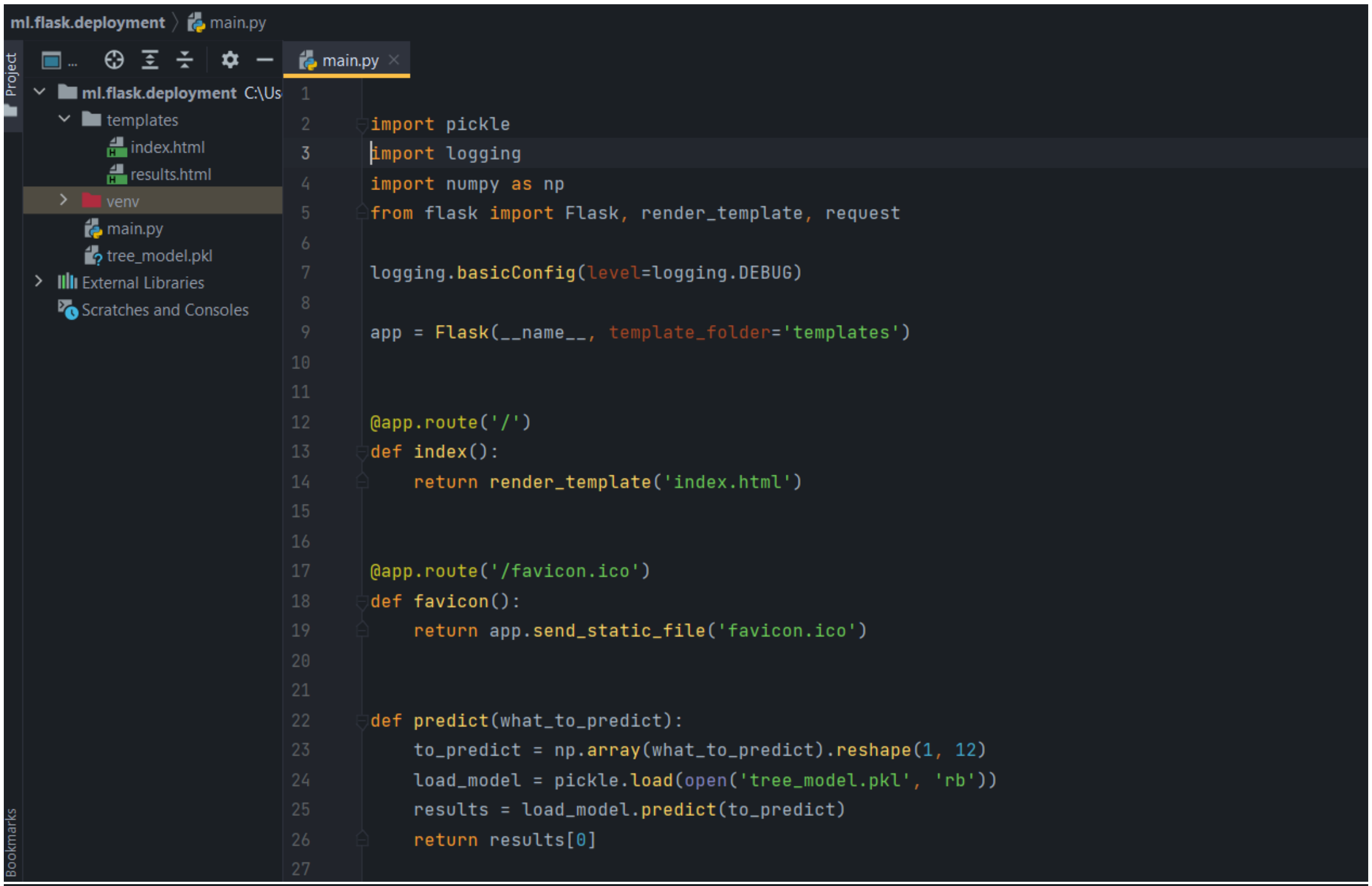
```python

# Serialize the model by using pickle
with open('tree_model.pkl', 'wb') as file:
    pickle.dump(tree_model, file)

# Save the dataframe that was used to train model
df.to_pickle('flask_model_df.pkl')
```



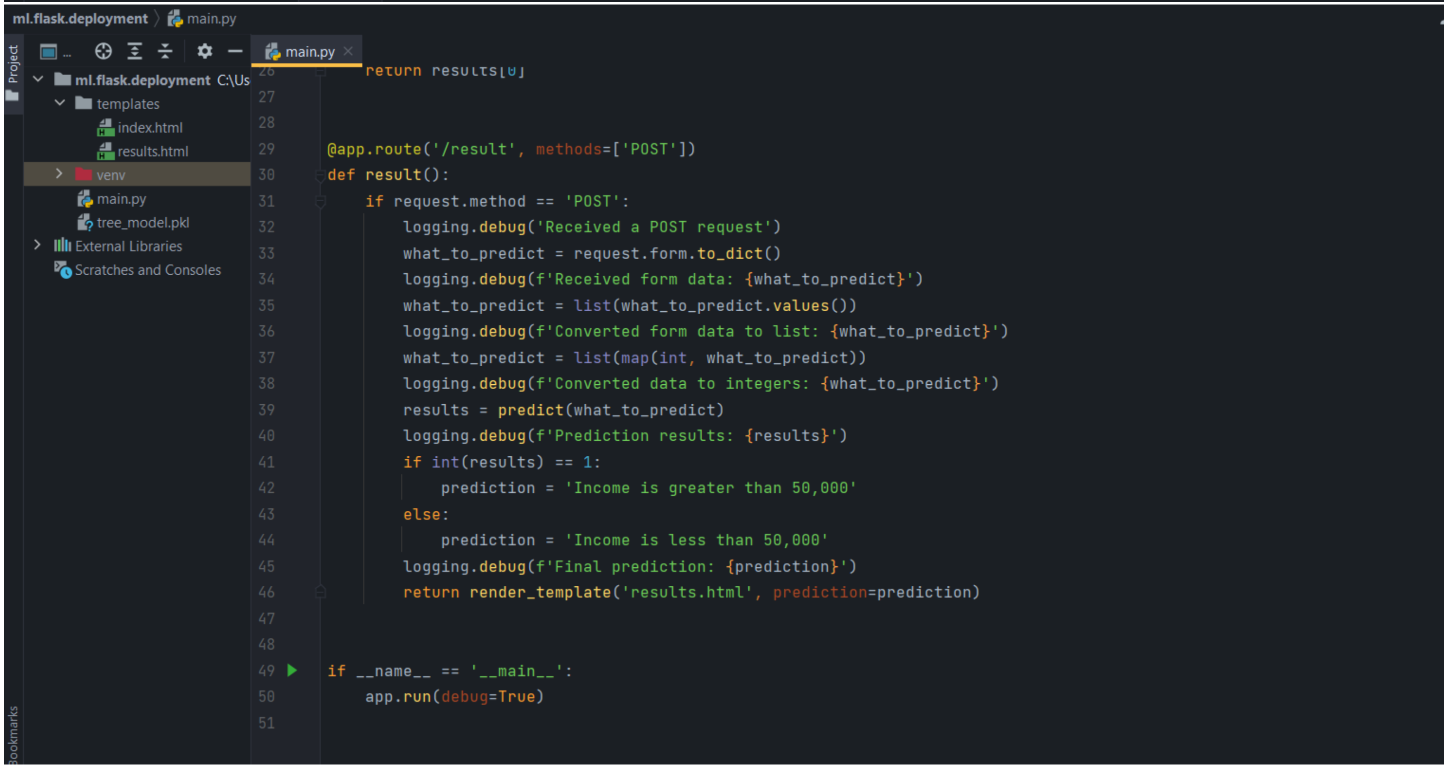
Switch To PyCharm for App Development

In [ ]:

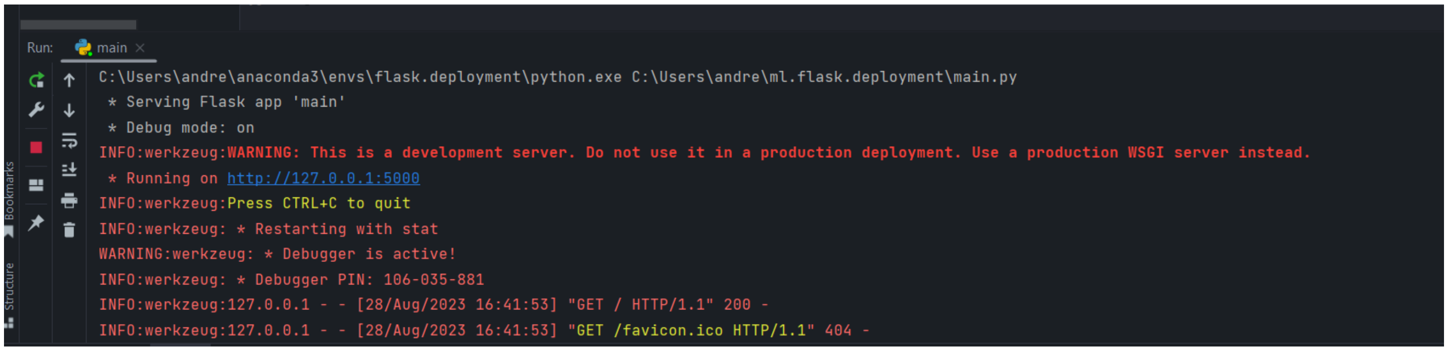
```python
# Please note that the point of this project is to demonstrate the building of a Flask web application
# Therefore, model accuracy, and model selection was not pertinent to this context
```



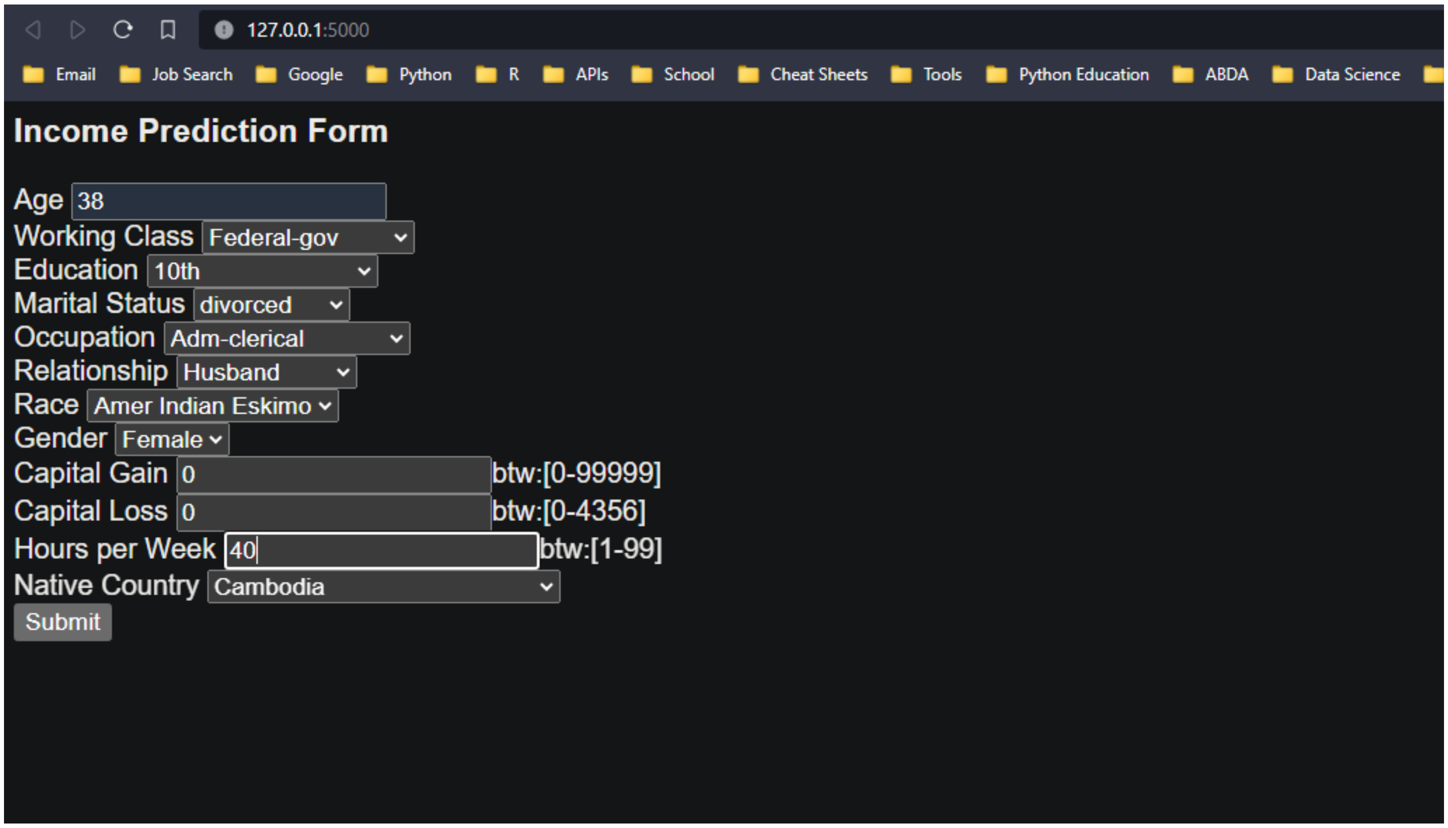
```python

import pickle
import logging
import numpy as np
from flask import Flask, render_template, request

logging.basicConfig(level=logging.DEBUG)

app = Flask(__name__, template_folder='templates')


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/favicon.ico')
def favicon():
    return app.send_static_file('favicon.ico')


def predict(what_to_predict):
    to_predict = np.array(what_to_predict).reshape(1, 12)
    load_model = pickle.load(open('tree_model.pkl', 'rb'))
    results = load_model.predict(to_predict)
    return results[0]

```



```python
    return results[0]


@app.route('/result', methods=['POST'])
def result():
    if request.method == 'POST':
        logging.debug('Received a POST request')
        what_to_predict = request.form.to_dict()
        logging.debug(f'Received form data: {what_to_predict}')
        what_to_predict = list(what_to_predict.values())
        logging.debug(f'Converted form data to list: {what_to_predict}')
        what_to_predict = list(map(int, what_to_predict))
        logging.debug(f'Converted data to integers: {what_to_predict}')
        results = predict(what_to_predict)
        logging.debug(f'Prediction results: {results}')
        if int(results) == 1:
            prediction = 'Income is greater than 50,000'
        else:
            prediction = 'Income is less than 50,000'
        logging.debug(f'Final prediction: {prediction}')
        return render_template('results.html', prediction=prediction)


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

```



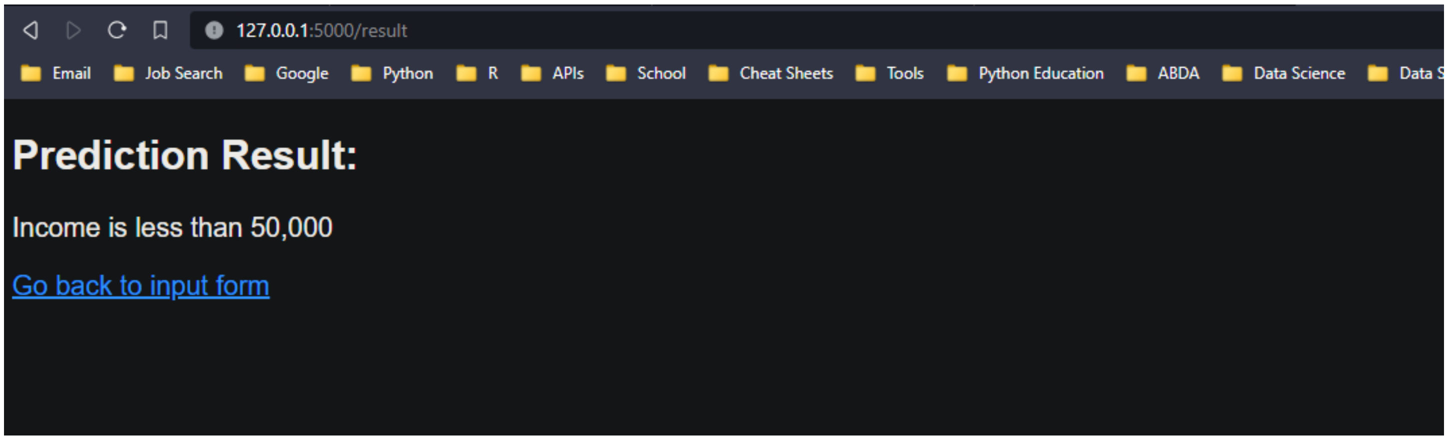
```
C:\Users\andre\anaconda3\envs\flask.deployment\python.exe C:\Users\andre\ml.flask.deployment\main.py
 * Serving Flask app 'main'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 106-035-881
INFO:werkzeug:127.0.0.1 - - [28/Aug/2023 16:41:53] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [28/Aug/2023 16:41:53] "GET /favicon.ico HTTP/1.1" 404 -
```



127.0.0.1:5000

Income Prediction Form

Age	38
Working Class	Federal-gov
Education	10th
Marital Status	divorced
Occupation	Adm-clerical
Relationship	Husband
Race	Amer Indian Eskimo
Gender	Female
Capital Gain	0 btw:[0-99999]
Capital Loss	0 btw:[0-4356]
Hours per Week	40 btw:[1-99]
Native Country	Cambodia



127.0.0.1:5000/result

Prediction Result:

Income is less than 50,000

Go back to input form